TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

SAMPO TOLVANEN

PÄÄSTÄ PÄÄHÄN -TESTAUS TUOTEKEHITYKSESSÄ

Diplomityö

Tarkastaja: Prof. Hannu-Matti Järvinen
Tarkastaja ja aihe hyväksytty
Tieto- ja sähkötekniikan tiedekunta-
neuvoston kokouksessa 31. touko-
kuuta 2017

TIIVISTELMÄ

SAMPO TOLVANEN: Päästä päähän -testaus tuotekehityksessä
Tampereen teknillinen yliopisto
Diplomityö, 45 sivua, 12 liitesivua
Toukokuu 2018
Tietotekniikan koulutusohjelma
Pääaine: Pervasive Systems
Tarkastajat: Prof. Hannu-Matti Järvinen
Avainsanat: Testaus, Testiautomaatio

Päästä päähän -testauksessa tutkitaan toimivatko kokonaisen sovelluksen sisältämät komponentit yhdessä niin kuin niiden on suunniteltu toimivan. Siinä testitapauksia suoritetaan kokonaisina käyttötapauksina. Päästä päähän -testausta voidaan käyttää testiautomaatiossa sen yhtenä testausmuotona muun testauksen kanssa. Tässä työssä tutkitaan päästä päähän -testausta ja testausautomaatiotyökaluja. Työn tavoitteena on tutkia voidaanko samoja testaustyökaluja käyttää useammassa ohjelmistotuotteessa. Olemassa olevien testaustyökalujen soveltuvuutta selvitettiin kahdelle ohjelmistotuoteprojektille. Testaustyökaluissa keskitytään web-sovelluksiin.

Työn toisessa osassa suoritettiin tutkimus, käyttäen muodostettua valintaprosessia. Prosessissa kartoitettiin saatavilla olevia päästä päähän -testaukseen soveltuvia työkaluja, joista pyrittiin löytämään parhaiten sopiva. Karsintaan muodostettiin lista kriteerejä, joilla kartoitettuja vaihtoehtoja saatiin vähennettyä pintapuolisella tarkastelulla alle puoleen alkuperäisestä määrästä. Työkalujen määrää saatiin vähennettyä sovelluskohteen kriteerien ja tarkennuksien avulla riittävän pieneksi tarkemman vertailun toteuttamiseksi. Jäljellä olevien työkalujen vertailun avulla löydettiin tutkittuihin ohjelmistotuoteprojekteihin sopiva testaustyökalu.

Testityökalujen valintaan luotua prosessia voidaan hyödyntää toisissa testaustyökalun valintaprojekteissa. Työssä esitettyä prosessia voidaan muokata valintaongelman mukaan. Kirjallisuusosassa koottua tietoa valintaprosesseista ja päästä päähän -testauksesta voidaan hyödyntää valinnan toteuttamisessa.

ABSTRACT

SAMPO TOLVANEN: End to end testing in product development
Tampere University of Technology
Diplomityö, 45 pages, 12 Appendix pages
May 2018
Master's Degree Programme in Information Technology
Major: Pervasive Systems
Examiner: Prof. Hannu-Matti Järvinen
Keywords: Testing, Test automation

End-to-end testing assesses that the flow of an application and the integrated components of an application are working as intended. The tests cases involve real use scenarios. With test automation, end-to-end-testing can be used as one way of testing along with other forms of testing. This thesis examines end-to-end testing and test automation tools. The goal is to find out if the same testing tools can be used in multiple software product development projects. The suitability of existing testing tools was evaluated against two different software products. The main area of focus is on web applications.

In the second part of the thesis a study is carried using the provided selection process. In the selection process some the of available alternatives suitable for end-to-end testing were explored in order to find a suitable option for the selection. A list of criteria was formed, which was used to reduce the number of alternatives to less than half of the original amount. Using the criteria for the software products and further adjustment of the requirements the amount of alternatives was reduced to be small enough for carrying out a tool comparison. A suitable alternative for the two investigated software products was found from the remaining tools.

The selection process created for tool selection can be made use of in other tool selection projects. The process can be modified according to the selection problem that is being solved. The information collected in the literature study about the selection processes and end-to-end testing can be used in carrying out a tool selection.

ALKUSANAT

Haluan kiittää työnantajaani Wapice Oy:tä diplomityöaiheesta. Haluan myös kiittää professori Hannu-Matti Järvistä työn ohjauksesta sen kirjoitusprosessin aikana.

Tampereella, 21.5.2018

Sampo Tolvanen

SISÄLTÖ

1. Johdanto 1

2. Tausta 3

2.1 Päästä päähän -testaus 3

2.2 Yleiset ongelmat 5

2.3 Tuotteet ja kriteerit 7

3. Tutkimusmenetelmät 9

3.1 Yleiset valintamenetelmät 9

3.2 Valintamenetelmät tuotteille 13

3.3 Tiedonhaun kriteerit 15

4. Valintaprosessi 17

4.1 Karsinta 17

4.2 Tuotetestaus 19

4.3 Työkalut 23

4.3.1 Katalon 23

4.3.2 Nightwatch.js 25

4.3.3 TestCafe 26

4.3.4 Robot Framework 28

5. Tulokset 30

5.1 Pisteytysperusteet 30

5.2 Pisteytys 31

6. Arviointi 38

7. Yhteenveto 41

Lähteet 43

Liite A. Testityökalujen kartoitus 46

KUVAT

3.1 MCDA:n yleiset vaiheet 12

3.2 Valinnan vaiheet 14

4.1 Katalon-työkalun käyttöliittymä 23

4.2 Robot Framework -työkalun RIDE-käyttöliittymä 28

TAULUKOT

4.1 Karsinnan koostetut tulokset 18

4.2 IoT-Ticketin 1. testitapauksen suorituskestot 20

4.3 IoT-Ticketin 2. testitapauksen suorituskestot 20

4.4 IoT-Ticketin 3. testitapauksen suorituskestot 20

4.5 IoT-Ticket-testiajojen kokonaissuoritusajat 21

4.6 EcoReactionin 1. testitapauksen suorituskestot 21

4.7 EcoReactionin 2. testitapauksen suorituskestot 21

4.8 EcoReactionin 3. testitapauksen suorituskestot 22

4.9 EcoReaction-testiajojen kokonaissuoritusajat 22

5.1 Testityökalujen pisteytys IoT-Ticketille 32

5.2 Testityökalujen pisteytys Ecoreaction-tuotteelle 32

LIST OF ABBREVIATIONS AND SYMBOLS

CD	Continuous Delivery
CI	Continuous Integration
DOM	Document Object Model
E2E	End-to-end
HTML	Hypertext Markup Language
MCDA	Multi-Criteria Decision Analysis
OTS	Off-the-shelf

1. JOHDANTO

Ohjelmistoja testataan useilla eri tavoilla kuten yksikkö-, integraatio-, systeemi- ja hyväksymistestauksella. E2E (end to end)-testauksessa, eli päästä päähän -testauksessa, pyritään varmistamaan, että sovelluksen sisältämät integroidut komponentit toimivat odotetulla tavalla [1]. Testaus tehdään todellisessa ympäristössä vastaten ohjelmiston oikeita käyttötapauksia. Ohjelmistojen testausta voidaan suorittaa testaten toiminnallisuutta, suorituskykyä ja ei-toiminnallisia ominaisuuksia kuten käytettävyyttä ja turvallisuutta.

Testausta voidaan tehdä käyttäen useanlaisia strategioita. Lasilaatikkotestauksessa tarkastellaan ohjelman sisäistä rakennetta, jonka pohjalta voidaan luoda testidataa [2, s. 11]. Mustalaatikkotestauksessa tutkittavaa järjestelmää tutkitaan sen spesifikaation mukana, eikä siinä hyödynnetä ohjelman sisäistä rakennetta [2, s. 9]. Harmaalaatikkotestauksessa yhdistellään näitä kahta lähestymistapaa. Päästä päähän -testausta voidaan tehdään mustalaatikkotestausta tai harmaalaatikkotestausta käyttäen.

Testausta voidaan tehdä automatisoidusti. Yksi merkittävä syy testiautomaation käyttöön on kustannustehokkuus. Testiautomaation kustannukset voivat olla alussa korkeita, mutta automatisoitujen testien suorittamisen kustannukset voivat olla pieniä tai mitättömiä. [5, s. 520]

Päästä päähän -testausta voidaan tehdä tuotekehityksessä osana jatkuvan integraatiota (engl. continuous integration, CI) ja jatkuvaa toimitusta (engl. continuous delivery, CD). Jatkuvassa integraatiossa pyritään siihen, että ohjelmisto toimii milloin tahansa [3, s. 55]. Jatkuvassa toimituksessa ohjelmistossa pyritään varmistamaan, että se voidaan luotettavasti julkaista mihin aikaan hyvänsä [4, s.50].

Tässä työssä tutkitaan päästä päähän -testausta tuotekehityksessä. Työssä selvitetään miten päästä päähän -testausta voidaan tehdä, mihin ongelmiin siinä voidaan törmätä ja millä tavalla sitä voidaan lähteä kehittämään ohjelmistotuoteprojekteissa käyttäen testaustyökaluja. Työssä keskitytään erityisesti toiminnallisuuden testaamiseen, testausautomaatioon ja siihen soveltuvien työkalujen valintaan. Työn

tutkimusosassa tarkastellaan ensin prosesseja testaustyökalujen valintaan ja arviointiin. Käytetyn prosessin avulla tutkitaan olemassa olevien työkalujen soveltuvuutta kahdelle tarkasteltavalle ohjelmistotuotteelle. Tavoitteena on selvittää, voidaanko samoja työkaluja käyttää useamman tarkasteltavan tuotteen testauksessa. Testaustyökaluissa keskitytään ensisijaisesti web-sovelluksiin.

Työssä käsitellään ensin yleisesti päästä päähän -testausta ja siihen liittyviä ongelmia. Tämän lisäksi siinä esitellään tutkittujen ohjelmistotuotteiden asettamia kriteerejä etsittäville testaustyökaluille. Seuraavaksi tutkitaan kuinka testaustyökalujen valinta voidaan toteuttaa. Se tehdään kahdessa vaiheessa, ensin ottamatta huomioon valittujen ohjelmistotuotteiden ominaisuuksia, jonka jälkeen tehdään tarkempi karsinta ne huomioiden. Tämän jälkeen kuvataan testaustyökalujen valintaprosessin eri vaiheiden toteutus. Tuloksissa pisteytetään vertailtavat työkalut ja analysoidaan saadut tulokset. Lopuksi työn onnistumista tutkitaan arviointiosuudessa.

2. TAUSTA

Tässä luvussa esitellään päästä päähän -testauksen piirteet testauksessa, yleiset ongelmat sekä vertailussa kokeiltavat ohjelmistotuotteet ja niiden asettamat kriteerit automatisoidulle päästä päähän -testaukselle.

2.1 Päästä päähän -testaus

Päästä päähän -testauksessa suoritetaan sovelluksen käyttötapauksia alusta loppuun. Testitapaukset voidaan liittää käyttötapauksien kautta järjestelmän vaatimuksiin ja niitä voidaan käyttää testauksen vaatimuskattavuuden tukena.

Testitapaukset suoritetaan ohjelmiston tarjoaman ulkoisen liitännän kautta, joka voi olla rajapinta tai graafinen käyttöliittymä. Tässä työssä keskitytään ensisijaisesti testaamiseen graafisen käyttöliittymän avulla. Sen erityispiirteitä testauksen kannalta ovat muun muassa käyttöliittymien korkea muuttuvuusaste, monimutkaisuus [5, s. 519], asynkronisuus sekä testitapauksien suoritusnopeus.

Testaustyökalut voivat suorittaa testejä tai ne voivat olla rakennettu helpottamaan manuaalista testausta. Käyttöliittymätestaustyökalut jakaantuvat nauhoitus- ja skriptipohjaisiin ratkaisuihin [5, s. 26, 65]. Nauhoituspohjaisessa testauksessa käyttäjä suorittaa käyttöliittymässä toimintoja, jotka tallennetaan ja tästä syntyy testitapaus, joka voidaan sen jälkeen suorittaa joko manuaalisesti tai automatisoidusti. Nauhoituksen heikkoutena on se, että nauhoituksesta syntyvä skripti on tiukasti sidottuna käyttöliittymän pieniin yksityiskohtiin [6, s. 104]. Tämä tulee ongelmaksi silloin, kun käyttöliittymään tulee muutoksia. Tällöin testitapaukset joudutaan mahdollisesti nauhoittamaan uudelleen.

Testitapauksia voidaan luoda käyttäen erilaisia menetelmiä. Skriptipohjaisessa testauksessa testin suoritusta ohjataan skriptin avulla. Avainsana-pohjaisessa testauksessa testitapaus kuvataan käyttäen avainsanoja ja mahdollisia parametrejä. Siinä on usein mahdollista luoda uusia avainsanoja skriptikieltä tai olemassa olevia avainsanoja käyttäen. Yksi tämän lähestymistavan eduista on, että testiskripteissä voidaan välttää monimutkainen logiikka ja se helpottaa ei-ohjelmoijia luomaan

automatisoituja testitapauksia [6, s. 115]. Mallipohjaisessa testauksessa tutkittavan järjestelmän pohjalta tehdään malli, jonka avulla voidaan luoda testitapauksia automatisoidusti [7, s. 14]. Lähestymistavan tarkoituksena on kattaa mahdollisimman hyvin sovelluksen suorituspolut. Mallipohjaisen testauksen lähestymistapaa voidaan käyttää päästä päähän -testauksessa, jos testitapauksien suorittamisesta ja niiden määrästä muodostuva suoritusaika ei nouse liian suureksi.

Web-sovelluksien päästä päähän -testaus suoritetaan jokaisella selaimella, joilla kehitettävän ohjelmiston halutaan varmistaa toimivan. Selaimia voidaan ajaa palvelinympäristöissä käyttäen selaimen ajon yhteydessä käyttäen näytön ulostuloa. Niitä voidaan ajaa myös headless-tilassa, mikä ei vaadi näytön käyttämistä. Erilaisia tapoja tarkastella tutkittavan web-sovelluksen tilaa ovat esimerkiksi DOM (engl. Document Object Model) -rakenteen lukeminen ja ruudunkaappausten analysointi. DOM:n tilaa käytetään paikantamaan verkkosivun elementtejä, joista voidaan hakea tietoja ja tai joihin voidaan kohdistaa toimintoja. Ruudunkaappauksessa saadusta kuvasta paikannetaan halutut elementit, joille voidaan tehdä tarkistuksia. Niistä voidaan esimerkiksi tarkastaa, että sivu tai sen osa pysyy ulkonäöltään muuttumattomana.

Päästä päähän -testauksen testisautomaatiota voidaan suorittaa testausta varten luoduilla palvelinkoneilla. Toinen mahdollisuus on ulkoistaa testausautomaatioon kuuluva infrastruktuuri sellaista tarjoavalle palveluntarjoajalle. Tässä lähestymistavassa testien suoritus tapahtuu palveluntarjoajan palvelimilla. Sen etuna on, että testattavaa palvelua tuottavan kehitystiimin ei tarvitse ylläpitää testausohjelmistoa eikä selainversioita. Yksi mahdollinen haittapuoli on se, että palveluntarjoaja voi kerätä kehitettävästä palvelusta tietoja. Lisäksi tarkan tiedon kerääminen on vaikeampaa verrattuna täysin itse hallittavissa oleviin palvelinkoneisiin. Tässä työssä testausta palveluna tarjoavat pilvipalvelut jätetään tarkastelun ulkopuolelle johtuen edellä esitetystä ongelmasta tiedon hallittavuuden kanssa.

Päästä päähän- ja käyttöliittymätestaukseen on luotu suunnittelumalleja. Yksi näistä on page object -suunnittelumalli, jossa verkkosivun elementtien käsittelyyn luodaan rajapinta, jotta HTML:ää (engl. Hypertext Markup Language) ei käsiteltäisi suoraan testitapauksissa [8]. Sitä voidaan käyttää myös muiden UI-teknologioiden kanssa. Suunnittelumalli parantaa koodin uudelleenkäyttöä ja mahdollistaa muokkauksien tekemisen testauksessa suoritettaviin toimintoihin, ilman testitapausskriptien muokkausta.

Testiautomaatiossa testitapauksia voidaan ajaa jatkuvan integraation ja jatkuvan toimituksen osana [3, s. 3-4]. Niissä koodimuutoksille tehdään ensin käännös, jonka

jälkeen suoritetaan testiajot. Päästä päähän -testaus voidaan suorittaa siinä osana hyväksymistestausta. Prosessi voidaan käynnistää jokaisen ohjelmakoodimuutoksen yhteydessä tai harvemmin. Tämän hyötyinä saadaan muun muassa parantunut näkyvyys, nopeampi palaute sekä parempi valmius julkistaa ohjelmaversio annetulla ajanhetkellä [3, s. 4]. Testiajoja voidaan tehdä harvemmin, mikäli ne ovat kestoltaan pitkiä.

Testiautomaatio helpottaa päästä päähän -testausta, mutta ei poista manuaalisen testauksen tarvetta. Manuaalista testausta tarvitaan esimerkiksi visuaalisten muutoksien varmentamisessa ja käytettävyystestauksessa. Käyttöliittymien testaukseen ja käytettävyystestaukseen sisältyy paljon tutkivaa testausta ja erilaisia suorituspolkuja. Uusien automaatiotestien luominen käyttöliittymän muuttuessa ei ole kustannustehokasta, mikäli niiden luonti on hidasta ja testitapauksia tarvitaan paljon.

2.2 Yleiset ongelmat

Testiautomaatiossa voidaan törmätä monenlaisiin ongelmiin, joiden takia testiautomaatio ei ole valmis testitapauksien luonnin jälkeen [5, s. 191]. Päästä päähän -testaukseen liittyy piirteitä, jotka lisäävät testiautomaatioon liittyviä haasteita.

Käyttöliittymätestit voivat olla hitaampia kuin muut alemman tason testit, kuten yksikkötestit ja integraatiotestit. Käyttöliittymän toimintojen suoritukseen kuuluu ohjelman muun toiminnan suorituksen lisäksi käyttöliittymän näkymän päivittäminen. Käyttöliittymätestit voivat olla suorituspituudeltaan pitkiä ja testien suuri määrä kasvattaa testiajojen pituutta. Tämä heikentää kykyä saada palautetta muutoksien aiheuttamista vaikutuksista. Yksittäisen testitapauksen suuri pituus hidastaa niissä ilmenevien ongelmien korjaamista, sillä usein testitapauksen suoritusta ei jatketa ensimmäisen kohdatun ongelman jälkeen. Käyttöliittymätestitapauksia voidaan jakaa kokoelmiin, joita ajetaan testauksen eri vaiheissa tai erilaisella aikatiheydellä. Yksi esimerkki tästä on savutestit, joissa pyritään varmistumaan siitä, että ohjelmiston uuden version julkaisu on valmistunut onnistuneesti [3, s. 117].

Testitapauksien tulokset voivat olla epäluotettavia antaen eri lopputuloksen, kun sama testi ajetaan testitapauksessa luodussa lähtötilanteessa [9, s. 3]. Epävakaat testitapaukset voivat toistuessaan saada kehittäjät jättämään testien epäonnistumisen huomiotta. Tämän käytännön laaja käyttö voi saada aikaan epäluottamuksen testiautomaatiota kohtaan. [9, s. 14]. Epävakaan testitapauksen epäonnistumisen syyn selvittämiseen voidaan menettää merkittävä määrä aikaa [9, s. 2]. Testitapauksien suorittamisen hitauden takia muutoksista saadaan hitaasti palautetta. Epäluotettavat tulokset ja hidas palaute yhdessä voivat aiheuttaa tilanteen, jossa epäonnistuvat

testit viivästyttävät ohjelmistoversion käyttöönottoa tai julkaisua. Roseberryn mukaan on yleistä, että osa testitapauksista jää epävakaaksi myös niiden vakauttamisen eteen tehtävän työn jälkeen [9, s. 9].

Epävakaat testitapaukset voivat johtua viallisista toteutuksista. Usein ongelmat esiintyvät testitapauksissa satunnaisesti vaihtelevien suoritusaikojen ja asynkronisyyden takia. Synkronointiongelmien syynä voi olla, että toiminto suoritetaan väärään aikaan [5, s. 244], jolloin sen suoritus ei tapahdu normaalisti. Tämä voi tapahtua muun muassa, jos synkronointiongelmien ratkaisemiseen on käytetty viivekomentoja, joissa suoritukseen lisätään vakiopituinen tauko [9, s. 7]. Testattavan sovelluksen suoritusnopeus voi vaihdella, minkä takia ohjelma on viivekomennon suorituksen valmistuttua väärässä tilassa. Testitapaukset voivat olla epävakaita myös muista syistä, kuten testiajoihin liittyvän infrastruktuurissa ilmenevän ongelman takia.

Testien ylläpito voi vaatia merkittävästi aikaa [5, s. 191]. Se voi vaatia erityisesti käyttöliittymätesteissä paljon työtä, sillä ne voivat rikkoutua testattavan ohjelmiston sisäisistä ja käyttöliittymään tehtävistä muutoksista. Verrattuna alemman tason testeihin, kuten yksikkötesteihin, käyttöliittymään tehdyt muutokset eivät tyypillisesti voi rikkoa ohjelmiston sisäistä logiikkaa testaavia testitapauksia. Ohjelmiston sisäiseen logiikkaan tehdyt muutokset voivat rikkoa käyttöliittymätestejä. Virheiden syyn selvittäminen voi kestää myös merkittävän määrän aikaa, jos testitapauksien suorituspolku on pitkä. Ongelmaa voidaan yrittää lieventää keräämällä tietoa testiajojen yhteydessä, kuten kuvankaappaus testitapauksien epäonnistumisajankohdissa. Lisäksi ylläpitotyötä voidaan helpottaa tekemällä testiajoja riittävän usein, jotta rikkoutuneet testitapaukset huomataan mahdollisimman ajoissa.

Käyttöliittymätestien kustannustehokkuus voidaan menettää, jos niiden potentiaaliset ylläpitokustannukset nousevat liian korkeiksi. Tämä voi tapahtua silloin, kun testitapauksia on paljon ja niiden rikkoutuessa joudutaan tekemään paljon korjauksia. Käyttöliittymätesteissä toiminnallisuutta ei voida välttämättä testata yhtä kattavasti kuin matalamman tason testeissä, koska testitapauksien suuri määrä kasvattaa testiajojen kokonaisaikaa ja keskimääräistä ylläpitotyöhön kuluvaa aikaa.

Testitapausten nauhoittaminen voi aiheuttaa ongelmia niiden ylläpidettävyydelle. Mikäli nauhoituksen tuloksena saatu ohjelmakoodi ei hyödynnä uudelleenkäyttöä, ohjelmien muutoksien kohdalla testitapauksia voidaan joutua muokkaamaan erikseen. Nauhoittamista voidaan käyttää työkaluna, joka nopeuttaa automatisoitujen testitapauksien luomista. Tällöin nauhoittimen tuottamaa ohjelmakoodia muokataan käyttämään vähemmän herkästi muuttuvia elementtitunnisteita sekä siinä parannetaan uudelleenkäyttöä. Monet testitapauksista tekevät todennäköisesti samoja

asioita useampaan kertaan, jolloin yhteiset osuudet voidaan jakaa jaetuksi ohjelmakoodiksi. Mikäli nauhoitustyökalu ei tue testitapauksien jakamista moduuleihin, se tulee tehdä nauhoituksen jälkeen erikseen.

Testeissä voidaan törmätä testattavuuden ongelmiin. Testattavuudella tarkoitetaan muun muassa näkyvyyttä ja hallittavuutta [6, s. 122-123]. Järjestelmän riittävä näkyvyys tulee varmistaa ennen kuin testausautomaatiota lähdetään rakentamaan. Web-sovelluksien kohdalla tämä tarkoittaa esimerkiksi sitä, että testattavan verkkosivun elementit ovat paikannettavissa yksikäsitteisesti ja että käyttöliittymän tilan tulee olla näkyvissä.

Käyttöliittymätestien automatisointi voi vaatia testien kehittäjältä erikoisosaamista. Käyttöliittymätestien automatisoinnissa on erityispiirteitä, kuten käyttöliittymien suuri todennäköisyys muuttua sekä käyttöliittymäkoodin monimutkaisuus [5, s. 519]. Käyttöliittymätestauksessa voidaan tarvita ylimääräistä laitteistoa sekä ohjelmistoja.

2.3 Tuotteet ja kriteerit

Seuraavat kriteerit liittyvät työssä testauksen kohteena oleviin ohjelmistotuotteisiin. Niitä käytetään työkalun valinnan karsinta- ja kokeiluvaihessa. Kriteereitä selvitettiin tutkimuksen ja haastatteluiden avulla.

IoT-Ticket on Wapice Oy:n kehittämä alusta datan keruuseen, raportointiin ja analytiikkaan [10]. Alustan avulla voidaan tehdä seurantaa ja etähallintaa. Käyttäjien on mahdollista luoda reaaliajassa päivittyviä käyttöliittymiä. IoT-Tickettiä voidaan käyttää web-sovelluksella, sen tarjoamien julkisten ohjelmointirajapintojen kautta, sekä siihen voidaan yhdistää laitteita. Web-sovellusta voidaan käyttää tietokoneilla, tableteilla sekä mobiililaitteilla. IoT-Ticketin web-sovelluksen testaukseen käytettävien testaustyökalujen osalta vaaditaan, että testitapauksissa pystytään käsittelemään web-sivun dynaamisesti muuttuvaa sisältöä. Sivustolla voidaan suorittaa toimintoja, joiden jälkeen tulos tulee näkyviin asynkronisesti ennalta määrittämättömässä ajassa. Testitapauksia tulee pystyä suorittamaan moderneilla selaimilla. Osa IoT-Ticketin testitapauksista vaatii, että testitapauksissa voidaan käyttää IoT-Ticketin tarjoamia ulkoisia rajapintoja. Tämä vaatii skriptien suorituksessa mahdollisuuden suorittaa HTTP-pyyntöjä.

EcoReaction on Wapice Oy:n kehittämä työkalu energiankulutuksen seurantaan, suunnitteluun sekä ennakointiin [11]. EcoReactionia käytetään web-sovelluksen kautta. Sitä voi käyttää tietokoneilla, tableteilla sekä mobiililaitteilla. Siinä tuetut selai-

met ovat Chrome, Internet Explorer, Edge ja Firefox. EcoReaction-tuotteen käyttöliittymä on suunniteltu toimimaan eri kokoisilla selaimilla ja käyttöliittymä muuttuu selaimen resoluution muuttuessa. Selaimen tulee olla säädettävissä riittävän suureksi, jotta testitapauksia voidaan laittaa ajoon erilaisille laitteille tarkoitetuilla tyyleillä.

Molempien ohjelmistotuotteiden käyttöliittymät ovat toteutettu Single Page Application -toteutustekniikalla, missä sivun sisältö vaihtuu dynaamisesti kokonaisien sivunlatauksien sijaan. Käyttöliittymien toteutuksissa käytetään JavaScript-skriptikieltä. Tuotteille rakennettavan E2E-testiautomaation tulee olla integroitavissa jatkuvan integraation järjestelmään. Testiajoista pitää olla mahdollista luoda tuloksena selkeälukuinen testiraportti.

3. TUTKIMUSMENETELMÄT

Tässä luvussa kuvataan ensin yleisesti, kuinka testaustyökalun valintaa voidaan lähteä suorittamaan, ja myöhemmin, kuinka menetelmiä sovelletaan työn soveltavassa osuudessa.

3.1 Yleiset valintamenetelmät

IEEE 1062 -standardissa tunnistetaan kolme tapaa hankkia ohjelmistoja, jotka ovat mukautetut, off-the-shelf (OTS) sekä software as a service (SaaS) -ohjelmistot [12, s. 13]. Mukautetuissa ohjelmistoissa ohjelmiston hankkija voi tuottaa sovelluksen tai sen tuottaminen voidaan ulkoistaa kolmannelle osapuolelle. Mukautettujen ohjelmistojen hyvänä puolena on muun muassa se, että ne saadaan sisältämään juuri se, mitä ohjelmistossa tarvitaan. Niissä saadaan joustavuutta ja hallittavuutta [12, s. 14]. Mukautettujen ohjelmistojen haittapuolena on, että ne ovat usein ratkaisuna kalliita ja niiden toteutus vie enemmän aikaa verrattuna muihin ratkaisuihin [12, s. 14]. Off-the-shelf -ratkaisuissa otetaan käyttöön olemassa oleva ratkaisu ohjelmistotoimittajalta. OTS-ratkaisujen etuina ovat niiden nopeus verrattuna mukautettuihin ohjelmistoihin ja niiden tarjoama hallittavuus datalle ja pääsynhallinnalle. OTS-ratkaisujen haittapuolia ovat muuan muassa ongelmat lisenssien kanssa, mahdolliset puutteet vaatimuksien kattavuudessa, hankaluudet ongelmien raportoimisessa, sekä se, että ohjelmiston päivittäminen ovat toimittajan hallinnassa.

E2E-testaukseen tarkoitettua yleisesti soveltuvaa ohjelmistoa on perustellusti hyvä hakea ensisijaisesti off-the-shelf -ratkaisuista. Mukautetun ratkaisun tulisi olla vaihtoehto vain tilanteessa, jossa valmiit ratkaisut eivät sovellu ratkaistavaan ongelmaan. Tämän perusteella tässä työssä tutkitaan voidaanko työkalu löytää olemassa olevien ratkaisujen joukosta. Ongelma parhaan työnkalun valinnasta luokitellaan valintaongelmaksi [13, s. 433]. Siinä yritetään pienentää lähtöjoukkoa niin pieneksi kuin on mahdollista saatavilla olevan tiedon avulla [13, s. 433].

Fewster ja Graham suosittelevat lähtemään työkalun valinnassa liikkeelle vaatimusten kartoituksesta [5, s. 249]. Se vähentää niiden työkalujen tutkimiseen käytettävää aikaa, joissa vaatimukset jäävät täyttymättä.

IEEE:n ISO/IEC 14102:2008 standardissa suositellaan haettavaksi seuraavia tietoja [14, s. 10]:

A) tuottajan yleiset tiedot kuten liiketoimintastrategia, saatavilla oleva tuki

B) tuottajan tuotekohtainen kehitysstrategia

C) työkalun kulut, sisältäen hinnan, ylläpidon, muutokset ja koulutuksen

D) laitteiston ja ohjelmiston vaatimukset työkalun käyttämiseen

E) laitteiston ja ohjelmiston vaatimukset tuettavan tuotteen käyttöön

F) vaadittu koulutus työkalun tehokkaaseen käyttöön

G) työkalun toiminnalliset kyvyt

H) työkalun metodologia ja elinkaaren tuki

I) työkalun kytkökset ulkoisiin järjestelmiin

J) käyttäjien määrä, käyttäjäryhmien olemassaolo ja käyttäjien vaste työkaluun

K) työkalun lisenssi (floating, multi-user, cross platform).

Tuottajan liiketoimintastrategia ja tuotekohtainen kehitysstrategia vaikuttavat päätöksentekoon työkalun käyttöiän kannalta. Työkalun muutostarpeiden määrän, käytön elinkaaren ja kehityselinkaaren tulee olla toistensa kanssa yhteensopivia, jotta sen käyttöä voidaan jatkaa sille suunnitellun elinkaaren ajan. Teknologian kehittyminen voi edellyttää, että työkalun tulee pysyä uuden teknologian kehityksen mukana [6, s. 107].

Kustannuksien arvioinnin merkittävänä tekijänä on kustannustehokkuus. Automatisoinnin tavoitteena on saavuttaa parempi kustannustehokkuus manuaaliseen työhön verrattuna. Testiautomaatiotyökalujen kustannukset koostuvat muuan muassa testiautomaation alkuperäisestä toteuttamisesta aiheutuvista kuluista, testiajojen kustannuksista sekä testiautomaation ylläpitokuluista.

Kaner et al. mainitsevat yhdeksi valintaperusteeksi työkalun tuttuuden [6, s. 107]. Tämä vaikuttaa työkalun kuluihin tarvittavan koulutuksen, opettelun ja tehokkuuden osalta. Työkalun omaksumisen lisäksi tuttuus helpottaa myös testaajan ja ohjelmistoprojektin muiden osanottajien välistä yhteistyötä. Tuttuudella voidaan viitata esimerkiksi työkalussa käytettävään ohjelmointikieleen ja siinä käytettäviin käsitteellisiin malleihin.

Toiminnallisia kykyjä mitataan organisaation vaatimuksiin ja tuettavan tuotteen tarpeisiin verraten. Työkalun toiminnallisia kykyjä ei todennäköisesti voida arvioida kokonaisuudessaan ennen kuin sitä on käytetty merkittävässä määrin. Niihin liittyviä vaatimuksia voi tulla lisää myöhemmin tuotekehityksen mukana. Vaatimuksia voidaan selvittää etukäteen tutkimalla suunniteltuja testitapauksia ja niissä esiintyviä toimintoja.

Työkalun kytkennät ulkoisiin järjestelmiin voivat liittyä esimerkiksi laadunhallintaan tai testausprosessissa oleviin muihin työkaluihin. Mikäli työkalu ei tue kytkentää haluttuihin järjestelmiin, voi olla mahdollista kehittää integraatio järjestelmien välille. Tämä vaikuttaa näin ollen myös työkalun kustannuksiin.

Käyttäjien määrä, käyttäjäryhmien olemassaolo ja käyttäjien vaste tarjoavat yhden keinon arvioida työkalun käyttöastetta ja sen toimivuutta. Työkalun käytön suosio parantaa mahdollisuuksia löytää osaajia, jotka tuntevat työkalun entuudestaan ja mahdollisesti tarjoavat lisää tietoa sen käytöstä.

Työkalun lisenssi vaikuttaa käytettävyyteen sekä kustannuksiin. Sen soveltuvuutta voidaan arvioida käyttöehtojen, kustannusten ja tarvittavien lisenssien määrän avulla. Lisenssin hinnan lisäksi sopivuuteen vaikuttavat sen rajoitukset. Lisensseissä, joissa käyttäjämäärä on rajoitettu, voidaan törmätä ristiriitaan käytettävyyden ja kustannustehokkuuden välillä. Rajoitetut määrät lisenssejä saattavat aiheuttaa vaikeuksia niiden jakamisessa useamman kehittäjän kesken, mikä voi vaatia ylimääräistä aikataulutusta. Toisaalta kalliimpi hinta voidaan nähdä perustelemattomaksi.

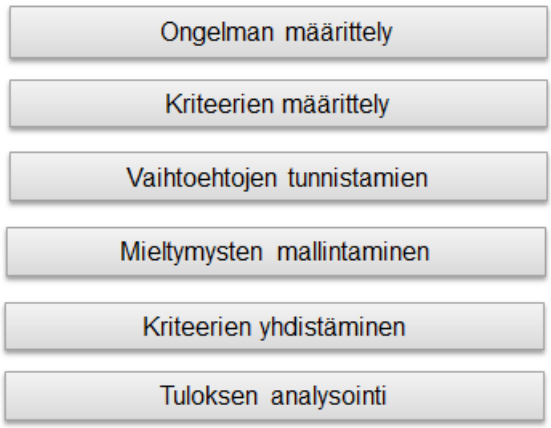
Fewster ja Graham mainitsevat laatukriteereissä dokumentaation laadun [5, s. 264]. Dokumentaation taso vaikuttaa työkalun käyttöön vaadittavaan koulutukseen ja sen tehokkaaseen käyttöön. Matalatasoinen dokumentaatio vaikeuttaa työkalun omaksumista ja sen itsenäistä käyttöä.

Kriteerien muodostamisen jälkeen voidaan edetä saatavilla olevien työkalujen kartoitukseen. Sen jälkeen saatavilla olevia työkaluja aletaan karsimaan [5, s. 270]. Kun vaihtoehtoja on saatu karsittua riittävän pieneen määrään, jäljellä olevia vaihtoehtoja voidaan vertailla tarkemmin kokeilemalla testaustyökaluja. Fewster ja Graham esittelevät mitattaviksi arvoiksi esimerkkeinä testitapauksien nauhoittamisen kestoa, niiden suorituksen kestoa verrattuna manuaaliseen ajoon sekä vertailun aikana huomattujen epäjohdonmukaisuuksien määrää [5, s. 278]. Testitapauksien luontiaika ei ole mitattavana arvona täysin luotettava, sillä niiden toteuttaminen yhdellä työkalulla voi nopeuttaa testitapauksien luontia seuraavilla työkaluilla. Toteutettavien testitapauksien tutkiminen voi vähentää tuttuuden vaikutusta toteutusaikoihin. Toinen tapa tehdä tuloksista paremmin vertailtavia on valita eri työkaluilla tehtä-

vät testitapaukset toteutettavaksi toiselle testaajalle, mutta tämän ongelmana on testaajien väliset nopeuserot. Vertailtavuuden parantamiseksi testitapauksien luonnissa pitäisi olla hyödyntämättä toisella testityökalulla luodun tapauksen rakennetta tai lähdekoodia.

Työkalujen kokeilemista varten tarvitaan testitapauksia. Fewster ja Graham ehdottavat testitapauksiksi yhden tavallisen testitapauksen ja yhden tavallista monimutkaisemman testitapauksen [5, s. 275]. Jälkimmäinen voidaan tunnistaa tutkimalla, mitkä asiat tekevät testitapauksen toteuttamisesta vaikeaa.

Kokeilemisen jälkeen valintaongelman ratkaisuun tarvitaan vertailumenetelmä, jota hyödyntämällä saadaan paras vaihtoehto. Päätöksenteoksen tueksi on saatavilla MCDA (engl. multi-criteria decision analysis) -menetelmiä. Nämä menetelmät tukevat päätöksentekoprosessia, johon sisältyy useita arvioitavia kriteereitä. MCDA-menetelmän valitsemista voi Ishizakan ja Nemeryn mukaan olla vaikea perustella, sillä mikään menetelmä ei ole täydellinen eikä ole soveltuva kaikkiin ongelmiin [15, s. 6]. Yksi yksinkertainen MCDA-menetelmä on painotettu summa. Tämän heikkoutena on vaikeus todeta painokertoimien oikeellisuus. Painotettua summaa käytettäessä kaikkien arvojen ei voida olettaa soveltuvan lineaariseen pisteytykseen. MCDA:n yleiset vaiheet ovat esitetty kuvassa 3.1, missä käytettiin apuna kirjallisuutta [16] [17].



Kuva 3.1 *MCDA:n yleiset vaiheet*

MCDA-prosessi sisältää paljon yhteisiä vaiheita Fewsterin ja Grahamin kuvaaman prosessin kanssa testityökalujen kartoitukseen ja valintaan. Fewster ja Graham suosittelevat vertailtavan muiden käyttäjien kokemuksia työkalusta [5, s. 278]. Käyttäjien kokemuksien avulla työkalusta voidaan saada tietoa, jota ei keritä saamaan rajoitetussa ajassa tehdyssä kokeilussa.

3.2 Valintamenetelmät tuotteille

Valintaprosessin ensimmäisessä vaiheessa etsitään testaustyökaluja arviointia varten ja niille tehdään karsinta. Olettaen, että saatavilla on paljon työkaluja, ensimmäisessä karsintavaiheessa käytetään kriteereitä, jotka voidaan tarkistaa nopeasti. Alkuperäinen testityökalujen etsintä tehdään tutkivan tiedonhaun keinoin. Tiedonhaussa käytetään Google-hakukonetta. Tuotteiden löytymiseen hakukoneesta vaikuttavat muun muassa työkalun käyttöaste sekä mainonta. Tiedonhaku tehdään tarpeen mukaan iteratiivisesti, hakemalla ensin riittäväksi todettu määrä erilaisia vaihtoehtoja. Mikäli näistä ei löydy sopivia vaihtoehtoja, hakuun voidaan palata myöhemmin uudelleen. Tuloksien monipuolisuuteen kiinnitetään huomiota hakemalla tietoa useammasta lähteestä ja tarkastelemalla löydettyjen tuloksien kattavuutta. Fewster ja Graham ehdottavat lähestymistavaksi olemassa olevien testaustyökalulistauksien käyttämisen [5, s. 270]. Google-hakukoneen yhteydessä maksetut hakutulokset jätetään huomiotta. Hakusanoina käytettiin lauseita "automated testing tools for web applications", ja "e2e testing framework".

Tiedonhaun ensimmäisessä vaiheessa keskitytään organisaatioita ja ohjelmistoja yleisesti koskeviin kriteereihin, mutta ei tarkastella ohjelmistotuotteita ja niiden vaatimuksia. Tämän perusteena on ajankäyttö, sillä vaihtoehtojen ja kriteerien määrän ollessa suuri alkuperäisten vaihtoehtojen selvittämiseen kuluisi merkittävästi enemmän aikaa verrattuna siihen, että karsintaa voidaan tehdä kriteereiden avulla nopeasti. Testaustyökaluista valitaan tarkasteluun ainoastaan ne, joilla on julkisesti ladattavissa oleva versio.

Tutkittavien työkalujen tulee soveltua funktionaliseen testaukseen. Yksinomaan muihin testauksen tyyppeihin, kuten kuormitustestaukseen, visuaaliseen testaukseen ja tietoturvatestaukseen tehdyt työkalut jätetään huomiotta.

Ostotilanteessa sitoudutaan ISO 9001 -standardin mukaan keräämään tietoja ostotilanteesta, joita ovat muun muassa tuotteen hyväksymisen vaatimukset, proseduurit, laitteisto, sekä laadunhallinnan järjestelmävaatimukset. Standardin mukaan ostovaatimusten sopivuus tulee varmistaa ennen yhteydenpitoa toimittajaan. [18, s. 10]

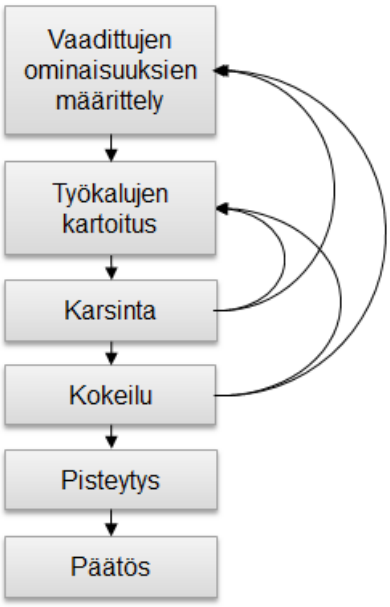
Työkaluvalintaprosessin tavoitteena on suorittaa valinta, joka johtaa työkalun onnistuneeseen käyttöön sen suunnitellun elinkaaren ajan. Valinnan kriteerien perusteena käytetään sitä, että niiden täyttämättömyys merkitsisi isoa riskiä työkalun onnistumiselle ja pitkäikäisyydelle. Jäljitettävyyden vuoksi hylättyjen työkalujen osalta dokumentoidaan hylkäämisen syy ja työkalun tarkastelun ajankohta.

Testaustyökalujen valinnassa arviointiin tulevat mukaan ohjelmistotuotteiden tes-

tauksen tarpeet ja tuotteen ominaisuudet. Projektien tarpeiden tunnistamiseksi käytetään testattavavien tuotteiden tutkimista sekä tarpeen mukaan apuna haastatteluita tai kyselyitä. Tuotteiden testaukseen liittyviä vaatimuksia selvitetään erikseen.

Testaustyökalujen ominaisuuksia käydään läpi verraten IoT-Ticket- ja EcoReaction-ohjelmistotuotteille asetettuihin vaatimuksiin. Pyrkimyksenä on löytää testityökaluja, jotka kattavat yksittäisen tuotteen vaatimukset ja mahdollisuuksien mukaan molempien ohjelmistotuotteiden vaatimukset. Karsinnassa pyritään rajaamaan saatavilla olevista vaihtoehdoista kolme parasta testaustyökalua. Karsinta tehdään jäljellä olevien työkalujen tarkempaan vertailuun kuluvan ajan rajaamiseksi.

Valinnan vaiheet esitetään kuvassa 3.2.



Kuva 3.2 *Valinnan vaiheet*

Karsinnan jälkeen testaustyökaluja kokeillaan toteuttamalla ja suorittamalla tuotteille laaditut testitapaukset. Mikäli testityökaluja ei saada karsittua riittävän pieneen lukumäärään, ensimmäisen ohjelmistotuotteen kohdalla käydään läpi jäljellä olevat testaustyökalut, jonka jälkeen tuotteiden kokeilua jatketaan niillä työkaluilla, jotka osoittautuvat ensimmäisen testauksen jälkeen lupaavimmiksi. Toimivuutta arvioidaan testauksesta saatujen tilastojen ja kriteerien täyttymisen suhteen. Työkalujen testaamisen jälkeen jälkeen suoritetaan valinta hyödyntäen päätöksentekomenetelmää. Vertailu tehdään eri testaustyökaluilla erikseen jokaista kokeiltavaa ohjelmistotuotetta varten. Tämän jälkeen tarkastellaan testityökalujen sopivuutta kaikkien kokeiltujen ohjelmistotuotteiden osalta. Pisteytyksessä käytetään MCDA-menetelmänä painotettua summaa.

3.3 Tiedonhaun kriteerit

Tarkasteltavana oleville ohjelmistotuotteille on yhteistä, että niitä käytetään web-käyttöliittymän avulla. Testaustyökaluja valitaan niin, että testitapauksia voidaan suorittaa web-selaimilla.

Testitapauksien suorituksen tulee olla automatisoitavissa. Työssä läpikäytävien tuotteiden kehityksessä ei haluta tehdä työtä käsin ja käyttää aikaa samojen testien uudelleenajoon, mikäli testitapausten suoritus on helposti automatisoitavissa. Testiautomaatio on yksi merkittävimmistä syistä testaustyökalun valintaan.

Testituloksia pitää pystyä raportoimaan. Se auttaa kehittäjiä saamaan palautetta muutoksista mahdollisimman nopeasti. ISO 9001 -standardissa mainitaan, että organisaation tulee ylläpitää asiakasvaatimuksien täyttymiseen liittyvää tietoa [18, s. 6]. Vaatimusten täyttymistä voidaan seurata testitapauksien avulla, joiden tilasta saadaan tietoa testausraporteista. Työkalussa tulee olla tuki tuottaa raportteja, jotka ovat koneellisesti luettavissa. Tämän perusteena on testityökalun kytkeminen muihin järjestelmiin.

Testityökalun tulee olla aktiivisessa kehityksessä. Uudet selainversiot voivat tuoda uusia ominaisuuksia tai sisältää muutoksia, jotka suoraan rikkovat testityökalun toiminnan. Tämän takia työkalu voidaan haluta vaihtaa kokonaan, mikäli ohjelmistotestaustuotteen kehitys on riittämätön, eikä kehitystä ei voida tai haluta ottaa omalle vastuulle. Myöhemmässä vaiheessa tutkittavien ohjelmistotuotteiden kehityselinkaaret ovat odotusarvoltaan pitkiä, minkä takia on perusteltua olettaa, että testaustyökalu tulee tarvitsemaan ohjelmistotuotteiden kehityselinkaaren aikana muutoksia. Tämän kriteerin arviointi tapahtuu tutkimalla miten usein työkalua päivitetään, milloin viimeisin päivitys on tehty, millä kapasiteetilla ja aktiivisuudella tuotteen ongelmiin puututaan ja kuinka aktiivinen tuotteen yhteisö on. Mikäli työkalulle ei ole saatavilla päivityshistoriaa, se hylätään. Vaadittuun päivitystiheyteen vaikuttaa muun muassa valittavaan työkaluun liittyvän teknologian kehittymisen nopeus. Web-selaimien nopean kehittymisen perusteella tässä työssä suurin sallittu aika viimeisimmästä päivityksestä on yksi vuosi.

Testityökalulla tulee olla aktiivisia käyttäjiä. Tätä voidaan arvioida tuotteeseen liittyvien Internet-julkaisujen avulla. Kaupallisen tuotteen kohdalla tätä voidaan arvioida myös asiakasreferenssien avulla. Esimerkkejä yhteisöjen kohtaamispaikoista ovat keskustelufoorumit ja sähköpostilistat. Tässä työssä aktiivisuutta arvioitaessa useiden kuukausien hiljaisuus tai viestien vastaamatta jääminen kertoo aktiivisuuden olevan matala. Tätä kriteeriä sovelletaan ensisijaisesti avoimen lähdekoodin tuotteiden kanssa, sillä kaupallisten tuotteiden kohdalla käyttöaktiivisuus ei välttämättä

näy julkisesti.

Testityökalun kehityksen takana tulee olla yritys tai avoimen lähdekoodin projektin tapauksessa useamman ihmisen muodostama kehitystiimi. Yrityksen tapauksessa yrityksen eliniän tulee olla merkittävä. Tuotteen tulee olla käyttövalmis, eli siitä pitää olla vähintään yksi vakaa julkaisu. Tämä kriteeri merkitään aktiivisuuden yhteyteen.

Tuotteeseen tulee maksullisten tuotteiden osalta olla saatavilla teknistä tukea ja avoimen lähdekoodin projektien tapauksessa siinä tulee olla mahdollisuus raportoida työkalussa ilmeneviä ohjelmistovirheitä, joita työkalun kehittäjät korjaavat. Avoimen lähdekoodin projekteissa käytetään usein julkista issue tracking -järjestelmää, jonka avulla ongelmiin puuttumisen nopeutta voidaan arvioida.

Kohdeorganisaatiossa käytetään yleisimmin Windows- ja Linux-pohjaisia käyttöjärjestelmiä kehityskäytössä. Palvelimissa käyttöjärjestelmä on yleisimmin Linux-pohjainen, mutta myös Windows on mahdollinen. Testaustyökalun tulee tukea ainakin Windows-käyttöjärjestelmää, sillä Windows on organisaatiossa kehityskäytössä yleisin.

Testityökalun dokumentaation tulee olla nähtävissä ilman tuotteen ostamista. Dokumentaation tulee sisältää riittävä ohjeistus, jotta työkalu voidaan ottaa käyttöön ja että sen avulla voidaan luoda testitapauksia itsenäisesti.

Testitapauksia tulee voida luoda ohjelmoimalla. Tämän tulee mahdollistaa koodin uudelleenkäyttö. Pelkällä nauhoittamisella ei useimmiten voida rakentaa hyvää testiautomaatiota, sillä käyttöliittymät voivat muuttua usein. Skriptikielen tulee olla standardi ohjelmointikieli. Tähän liittyy tuotekohtaisen ohjelmointikielen opettelun vaikeus, testaajien ja ohjelmoijien yhteistyön vaikeutuminen sekä muiden tekemän työn käyttö [6, s. 118]. Tuotekohtaisen ohjelmointikielen käyttö vaikeuttaa osaajien löytymistä sekä voi vaikuttaa työnhakijoiden arvioon tarjotuista työtehtävistä, sillä sen hyödyntämisestä opitut taidot eivät ole yhtä hyvin sovellettavissa uusiin tehtäviin kuin yleisesti käytetyn teknologian tapauksessa.

4. VALINTAPROSESSI

Tässä luvussa kuvataan työkalun valinnan vaiheiden suoritus. Ensin esitellään tiedonhaun avulla saadut tulokset työkaluista, jotka kattavat karsinnan ensimmäiseen osaan kriteerit. Tämän jälkeen kuvataan karsintaprosessi, jonka avulla työkalujen määrä vähennetään riittävän pieneksi. Viimeiseksi esitellään jäljelle jääneet työkalut ja niille suoritettava testaus.

4.1 Karsinta

Alustavassa tiedonhaussa löydettiin yhteensä 26 web-testaamiseen soveltuvaa testaustyökalua, joilla voidaan suorittaa päästä päähän -testausta. Löydetyistä työkaluista 12 täytti alustavaan karsintaan asetetut kriteerit ja 14 työkalua hylättiin kriteereiden jäämättä täyttymättä. Tapaukset, joissa kriteerin täyttyminen oli epäselvää, on merkitty hylätyiksi.

Tiedonhaun tarkemmat tulokset ovat dokumentoitu liitteessä A. Koostetut tulokset ovat esitetty taulukossa 4.1. Hyväksytyt työkalut ovat merkitty vihreällä taustavärillä. Tuloksista nähdään että eniten karsivaksi kriteeriksi osoittautui skriptituki. Seuraavaksi eniten vaikutti työkalujen aktiivisuus.

Tämän jälkeen karsintaa jatkettiin tiukentamalla kriteerejä. Selaintuen vaatimuksia kovennettiin niin, että testaustyökalun tulee tukea ainakin Internet Explorer-, Firefox-, Chrome- ja Edge-selaimia. Tämä oli perusteltua käyttäen tutkittujen ohjelmistotuotteiden tukemia selaimia.

Seuraavaksi jäljelle jääneitä vaihtoehtoja tutkittiin tarkemmin. Squish-työkalusta huomattiin, että se on merkittävästi kalliimpi kuin muut vaihtoehdot, mutta sen tarjoamat toiminnallisuudet eivät ole työkalun verkkosivujen mukaan merkittävästi parempia kuin muissa vaihtoehdoissa. Tästä syystä Squish-työkalun merkittävää hintaa ei nähty perustelluksi ja työkalu jätettiin vaihtoehtojen ulkopuolelle. Sahi Pro on myös merkittävästi kalliimpi verrattuna jäljelle jääviin vaihtoehtoihin eivätkä sen tarjoamat ominaisuudet eroa paljoa muista työkaluista.

	Skriptituki	Selaintuki	Käyttöjärjestelmät	Raportointi	Aktiivisuus	Käytön tuki	Dokumentaatio
Qf-test	✓	✓	✓	✓	✓	✓	✓
Ranorex	✓	✓	✓	✓	✓	✓	✓
Sahi	✓	✓	✓	✓	✗	✓	✗
Sahi Pro	✓	✓	✓	✓	✓	✓	✓
Eggplant	✗	✓	✓	✓	✓	✓	✓
iMacros	✓	✓	✓	✗	✓	✓	✓
Selenium IDE	✗	✓	✓	✓	✓	✓	✓
TestComplete	✓	✓	✓	✓	✓	✓	✓
Telerik Test Studio	✓	✓	✓	✓	✓	✓	✓
Tricentis Tosca	✗	✓	✓	✓	✓	✓	✓
Watir	✓	✓	✓	✓	✓	✓	✓
Katalon	✓	✓	✓	✓	✓	✓	✓
TestCafe	✓	✓	✓	✓	✓	✓	✓
Unified Functional Testing	✗	✓	✓	✓	✓	✓	✓
TestingWhiz	✗	✓	✓	✓	✗	✓	✓
Serenity	✓	✓	✓	✓	✗	✓	✓
Robot Framework	✓	✓	✓	✓	✓	✓	✓
Squish	✓	✓	✓	✓	✓	✓	✓
Zaptest	✗	✓	✓	✗	✗	✓	✗
Heliumhq	✓	✓	✓	✗	✗	✓	✓
Canoo webtest	✗	✗	✓	✗	✗	✗	✗
Nightwatch.js	✓	✓	✓	✓	✓	✓	✓
Protractor	✓	✓	✓	✓	✓	✓	✓
Buster.js	✓	✓	✓	✓	✗	✗	✓
Appperfect webtest	✓	✓	✓	✓	✗	✓	✓
Redwood HQ	✓	✓	✓	✓	✗	✓	✓

Taulukko 4.1 *Karsinnan koostetut tulokset*

Tämän jälkeen testityökaluja tutkittiin kokeilemalla niiden käyttöönottoa. Vaiheen tavoitteena oli selvittää työkalujen alustava toimivuus. Tällöin kokeiltiin asennuksen ja testiajojen toimivuus useammalla eri selaimella. Testityökalujen asennuksen jälkeen niillä ajettiin yksinkertainen tutkittaviin ohjelmistotuotteisiin liittymätön testitapaus.

Kokeilussa ei ilmennyt ongelmia, joiden takia työkaluja olisi ollut syytä sivuuttaa. Vertailuun valittiin testityökaluiksi Katalon, TestCafe, Robot Framework ja Nightwatch.js.

4.2 Tuotetestaus

Testitapauksiksi valittiin erikseen lyhyt, ohjelmistotuotteen suhteen keskimääräinen sekä teknisesti haastava testitapaus. Jälkimmäisiin tapauksiin valittiin testitapauksia, joihin liittyi tavallista enemmän tarvetta niiden suorituksen synkronointiin. Jokainen testitapaus ajettiin läpi 10 kertaa, millä kokeiltiin testiajojen vakautta. Tämän yhteydessä mitattiin myös testiajojen pituudet. Testauksessa kokeiltiin myös syöttää testitapaukseen tarkoituksella virhe, jonka avulla nähdään kuinka helppoa testiajoissa ilmenneiden virheiden syitä on löytää. Testaustyökalun dokumentaatiota arvioitiin tapauksien toteuttamisen yhteydessä.

Testitapauksien luonnissa huomioitiin hyvinä käytäntöinä, että niissä käytettiin implisiittisiä tai eksplisiittisiä odotuksia viivekomentojen sijaan. Elementtien paikantamisessa käytettiin mahdollisuuksien mukaan semanttista tapaa paikantaa elementti, jonka tarkoituksena on yrittää pitää elementtien paikannus toimivana myös monien käyttöliittymämuutoksien jälkeen.

IoT-Ticket

Testitapaus 1

Testitapauksessa suoritettiin yksinkertainen toiminto, missä järjestelmään kirjauduttiin sisään ja tämän jälkeen järjestelmästä kirjauduttiin ulos. Tapauksen suoritus oli nopeaa eikä siinä törmätty ongelmiin. Testitapauksen suorituskestot esitellään taulukossa 4.2.

Testitapaus 2

Tässä suoritettiin kirjautuminen, navigointi toimintosivulle, toiminnon suorittaminen ja toiminnon lopputuloksen varmistaminen. Testitapauksen suorituskestot esitellään taulukossa 4.3.

	Suorituskesto (keskiarvo)
Katalon	7 s
Nightwatch.js	8 s
TestCafe	11 s
Robot Framework	10 s

Taulukko 4.2 *IoT-Ticketin 1. testitapauksen suorituskestot*

	Suorituskesto (keskiarvo)
Katalon	16 s
Nightwatch.js	18 s
TestCafe	18 s
Robot Framework	14 s

Taulukko 4.3 *IoT-Ticketin 2. testitapauksen suorituskestot*

Testitapaus 3

Tässä suoritettiin kirjautuminen, navigointi toimintosivulle, toiminnon suorittaminen ja kolmannen osapuolen järjestelmän kytkentä. Tämän jälkeen lopputulos varmistettiin asynkronisella odotuksella tarkastelemalla sivun reaaliaikaisesti muuttuvaa sisältöä. Muissa työkaluissa paitsi TestCafessa muuttuvan sisällön tarkasteluun tarvittiin mukautettu toiminto, millä oli mahdollista välttää Seleniumin Stale Element -virhetilanne. Testitapauksen suorituskestot esitellään taulukossa 4.4.

Testiajojen kokonaissuoritusaikojen keskiarvot ovat esitetty taulukossa 4.5. Testien suorituksessa nähtiin, että eri työkalut koostivat nämä tulokset eri tavoin, minkä takia lukuja ei voida verrata suoraan keskenään.

EcoReaction

Testitapaus 1

Testitapauksessa kirjauduttiin järjestelmään sisään ja tämän jälkeen järjestelmästä

	Suorituskesto (keskiarvo)
Katalon	19 s
Nightwatch.js	17 s
TestCafe	23 s
Robot Framework	20 s

Taulukko 4.4 *IoT-Ticketin 3. testitapauksen suorituskestot*

Työkalu	Suoritusajan keskiarvo
Katalon	50 s
Nightwatch.js	40 s
TestCafe	47 s
Robot Framework	50 s

Taulukko 4.5 *IoT-Ticket-testiajojen kokonaissuoritusajat*

	Suorituskesto (keskiarvo)
Katalon	9 s
Nightwatch.js	9 s
TestCafe	9 s
Robot Framework	9 s

Taulukko 4.6 *EcoReactionin 1. testitapauksen suorituskestot*

kirjauduttiin ulos. Tapaus saatiin suoritettua kaikilla työkaluilla ongelmitta. Sen suorituskestot esitellään taulukossa 4.6.

Testitapaus 2

Testitapauksessa suoritettiin kirjautuminen, navigointi kohdenäkymään, uuden entiteetin lisäys, entiteetin löytäminen käyttämällä hakutoimintoa sekä sen poistaminen. Testitapauksen suorituskestot esitellään taulukossa 4.7.

Testitapaus 3

Testitapauksessa suoritettiin kirjautuminen, navigointi kohdenäkymään, asetuksen muutos, muutoksen tallentamisen varmistaminen ja asetuksen palauttaminen sen alkuperäiseen arvoon. Katalon-työkalun raahaus ja pudotus -toiminto ei toiminut suoraan, vaan testitapauksessa jouduttiin käyttämään Seleniumin tarjoamaa rajapintaa. Lisäksi huomattiin, että joidenkin komentojen suoritus ei toiminut samalla tavalla Selenium-pohjaisissa työkaluissa eri selaimilla. Testitapauksen suorituskestot esitellään taulukossa 4.8.

	Suorituskesto (keskiarvo)
Katalon	14 s
Nightwatch.js	14 s
TestCafe	12 s
Robot Framework	14 s

Taulukko 4.7 *EcoReactionin 2. testitapauksen suorituskestot*

	Suorituskesto (keskiarvo)
Katalon	16 s
Nightwatch.js	13 s
TestCafe	12 s
Robot Framework	13 s

Taulukko 4.8 *EcoReactionin 3. testitapauksen suorituskestot*

Työkalu	Suoritusajan keskiarvo
Katalon	40 s
Nightwatch.js	36 s
TestCafe	34 s
Robot Framework	46 s

Taulukko 4.9 *EcoReaction-testiajojen kokonaissuoritusajat*

Testiajojen kokonaissuoritusaikojen keskiarvot ovat esitetty taulukossa 4.9.

Suoritusaikojen mittaukseen tehdyissä testiajoissa käytettiin Chrome-selainta. Suoritusajat eivät ole kaikkien työkalujen osalta vertailtavissa. Katalon-, Nightwatch.js- ja Robot Framework -työkaluilla toteutetuissa testitapauksissa näiden alussa käynnistetään selain, josta suoritusajan mittaus alkaa. TestCafe-työkalu ei käynnistä selainta uudelleen testitapauksien välissä. Katalon mahdollistaa testaajan päättää suoritustavan itse, joko käynnistäen selaimen ennen jokaista testitapausta tai käyttäen testitapauksissa samaa instanssia selaimesta.

Kokonaisaikaan vaikuttaa keskeisesti suurimman mahdollisen testisuoritusnopeuden lisäksi toimintojen synkronointi ja testien välissä tapahtuvat toiminnot. Mikäli testitapauksien välissä joudutaan toistamaan toimintoja, kuten selaimen uudelleenavaus tai ohjelman käyttöönotossa tapahtuva kirjautuminen, ne vaikuttavat testiajon kokonaisaikaan merkittävästi.

Suorituskykymittauksissa lähdettiin ensin liikeelle vertailemalla suoritusaikoja toisiinsa. Tämä avulla voitiin arvioida mitkä testitapaustoteutukset mahdollisesti vaativat suorituskykyparannuksia. Suorituskykyä voitiin tutkia parhaiten Katalon- ja Robot Framework -työkaluissa, joiden raporteissa on tallennettuna testitapauksen jokaisen askeleen suoritusaika. Suoritusaikojen perusteella kaikki vertaillut työkalut olivat suorituskyvyltään samankaltaisia. Katalon-, Nightwatch.js- ja Robot Framework -työkaluissa suorituskykyyn saatiin vaikutettua eniten parantamalla ekspliittisiin odotuksiin luotujen funktioiden nopeutta. Testauksessa huomattiin, että mittaukset on hyvä aloittaa, kun luotujen testitapauksien vakautta on saatu testat-

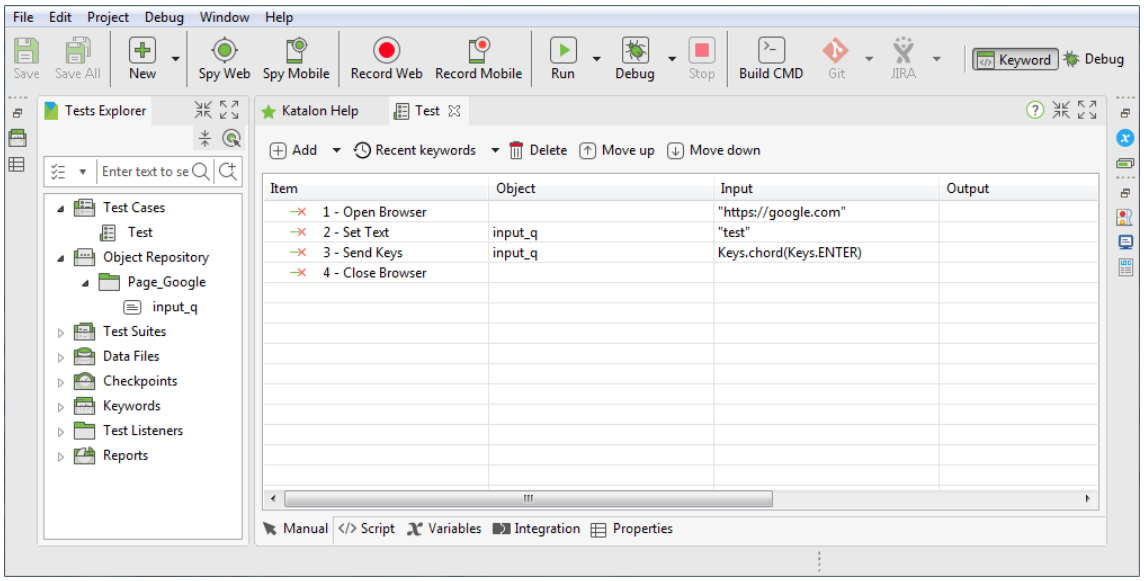
tua. Tällä tavoin voidaan vähentää todennäköisyyttä ongelman ilmenemiselle kesken suorituskykytestauksen, minkä korjaaminen vaatii testitapauksien muokkaamista ja mittauksien tekemisen uudelleen.

4.3 Työkalut

Tässä luvussa esitellään työkalujen valinnan kokeiluosuudessa tutkitut työkalut. Lisäksi esitellään testauksessa tehdyt havainnot.

4.3.1 Katalon

Katalon-työkalun testauksessa käytettiin versiota 5.3. Sen asennus vaatii Katalon Studio -ohjelman asennuksen, jonka jälkeen työkalu on täysin käytettävissä. Erillisiä asennuksia ei vaadita.



Kuva 4.1 *Katalon-työkalun käyttöliittymä*

Katalon-työkalussa testitapauksia voi luoda käyttäen nauhoitustyökalua, manual-näkymän graafista käyttöliittymää ja skriptinäkymää. Manual-näkymä tarjoaa käyttäjälle mahdollisuuden lisätä avainsanoja ja muokata niille annettavia parametrejä strukturoidusti. Pudotusvalikoiden kautta valittavat komennot ja parametrit ovat ryhmitelty ja rajattu, mikä helpottaa vaihtoehtojen etsimistä. Testitapaukset muodostuvat skriptistä, joka useimmiten koostuu peräkkäin suoritettavista komennoista. Skripteissä käytetään Katalonin tarjoamia rajapintoja. Kolmannen osapuolen ohjelmakoodia voidaan sisällyttää lisäämällä riippuvuuksiksi JAR-paketteja.

Nauhoitustyökalu osoittautui hyödylliseksi web-elementtien poimintaan. Siinä vältytään osalta nauhoitukseen liittyvistä ongelmista, koska sen aikana luodut elementit tallennetaan erikseen, eivätkä ne jää testitapauksiin kovakoodatuksi. Poimituille elementeille luetellaan kaikki tunnistetut ominaisuudet, joita testaaja voi käyttää muodostaakseen elementin hakuehdot. Testitapausten luonnissa huomattiin, että nauhoitintyökalun automaattisesti valitsemat ominaisuudet elementin tunnistamiseen eivät aina toimi suoraan.

Nauhoitustyökalulla ei useimmissa tapauksissa voi luoda testitapauksia suoraan, sillä nauhoituksessa luodut avainsanat eivät ota huomioon elementtejä, jotka luodaan tai tulevat näkyviin vaihtelevassa ajassa. Tämä vaatii avainsanojen lisäyksen, jolla viiveet huomioidaan tai avainsanojen vaihtamisen versioihin, joissa odotukset huomioidaan sisäisesti. Mikäli testitapauksia pitää voida luoda ilman skriptausta, tähän tarvitaan todennäköisesti yhteistyötä testiautomaatio-osaajan kanssa. Siinä testitapauksien luomisen helpottamiseksi voidaan luoda yleiskäyttöisiä avainsanoja. Yksi huomatuista puutteista oli, että nauhoitus ei kaapannut navigaatioon käytettäviä näppäinpainalluksia. Tämä vaatii näppäinpainalluskomentojen lisäyksen käyttöliittymän tai ohjelmakoodin kautta manuaalisesti.

Katalonin avainsanoja käytettäessä käyttöliittymän tulee olla oikeanlaisessa tilassa. Esimerkiksi kahden peräkkäisen toimintoavainsanan välissä voi tapahtua virhe, mikäli toinen avainsana suoritetaan ennen kuin tilanvaihdos päättyy. Asynkronisuuden huomioonottavia komentoja ei lisätä automaattisesti testiskriptiin testinauhoitusta käytettäessä. Tapauksissa, joissa sivun sisältö voi vaihtua käyttäjän navigoinnista riippumattomien tapahtumien seurauksena, voidaan törmätä Selenium-testeissä esiintyvään Stale Element -ongelmaan [19]. Siinä viite tarkisteltavaan elementtiin on vanhentunut, minkä takia annettua toimintoa ei voida suorittaa. Ongelma voi esiintyä tapauksissa, joissa elementti poistetaan DOM-rakenteesta kahden Selenium-komennon välissä [19]. Silloin, kun tilan muutoksia tapahtuu tarkoituksetta, syynä voidaan pitää myös virhettä ohjelmassa. Ennustamattomien tilanmuutosten tapauksessa ongelman voi ratkaista ohjelmoimalla toiminnon, jossa virhe jätetään huomiotta ja toimintoa yritetään uudelleen.

Katalon-työkalun käytössä uusien testitapauksien luontiin tarvitaan käytännössä Katalon Studio -käyttöliittymä, sillä Katalon-projektin rakenne koostuu prosessoiduista tiedostoista, joita on hankala luoda käsin. Katalon Studio ei tukenut kirjoitushetkellä Linux-käyttöjärjestelmää, minkä takia testitapauksia voidaan luoda siinä vain Windows- ja MacOS-käyttöjärjestelmissä.

Katalon tuottaa oletuksena sekä koneellisesti luettavan testiraportin että kehittäjän

käytössä olevan HTML-raportin. Katalon-ohjelmasta saadaan tarvittaessa kytkettyä päälle ominaisuus, joka tallentaa kuvankaappaukset testitapauksien epäonnistuessa ja lisää ne testiraporttiin. Testiajojen tulokset voi myös lähettää suoraan testitapauksien hallintaan tarkoitetulle qtest-työkalulle.

Katalon-työkalun dokumentaatio koostuu sen verkkosivuilta löytyvistä tutoriaaleista, ohjesivustosta, videoista sekä ohjelmointirajapinnan dokumentaatiosta. Näiden lisäksi Katalonille on saatavilla esimerkkiprojekteja. Työkalun käytön voi opetella itsenäisesti käyttäen sitä varten laadittua dokumentaatiota. Katalon-ohjelman käyttöliittymän ja nauhoitustyökalun tarjoamat toiminnot testitapausskriptin komentojen luontiin helpottivat testiskripteihin tutustumista. Katalon-työkalu tarjoaa runsaasti toimintoja, joiden hyödyntämisessä testaajalla tulee olla hyvä käsitys testiautomaation hyvistä käytännöistä.

4.3.2 Nightwatch.js

Nightwatch.js-työkalun testauksessa käytettiin versiota 0.9.19. Asennusta varten käytössä tulee olla Node.js-ympäristö sekä npm-paketinhallinta. Jokaiselle selaimelle tulee ladata ajuri, jolle tulee määrittää polku path-muuttujaan tai sen polku pitää määrittää käsin. Saatavilla on kolmannen osapuolen moduuleja, joilla lataus voidaan tehdä automaattisesti ja polku saadaan suoraan konfiguraatioihin.

Nightwatch.js-työkalussa testitapauksia luodaan kirjoittamalla testitapaukset JavaScript-skriptikielellä. Sen käyttöön tarvitaan testiautomaatio-osaaja. Skriptit ovat vapaasti laajennettavissa kaikilla saatavilla olevilla JavaScript-kirjastoilla. Testitapauksien luonnissa käytetään page object -suunnittelumallia. Sivuobjekteihin sisällytetään sivuilla käytettävien elementtien tiedot. Testitapauksien kehittäjällä tulee olla tietämystä verkkosivujen toiminnasta hakeakseen näitä tietoja. Nightwatch.js-työkalulle toteutetusta testitapauksesta löytyy koodilistauksesta 4.1.

```
module.exports = {
        'Example test case': function(browser) {
                var page = browser.page.testPage();
                page
                  .open()
                  .inputText("test")
                  .clickSubmit()
                browser.end();
        }
```

```
};
```

Ohjelma 4.1 *Esimerkki Nightwatch.js-työkalulla toteutetusta testitapauksesta*

Testitapauksien kehittäjän tulee etsiä elementtien tiedot ja sisällyttää ne sivuobjektien elementtien kuvaukseen. Yksi tapa tehdä tämä on käyttää web-selaimen kehittäjätyökaluja, joilla on mahdollista tutkia ruudulla näkyvien elementtien tietoja. Nightwatch.js-työkalun käyttämiseen tehokkaasti tulee ensin tutustua kuinka testitapauksia toteutetaan, jotta niistä saadaan hyvin ylläpidettäviä ja vakaita.

Testitapauksien suorituksen asynkronisuuden käsittelyssä käytetään odotuskomentoja, joiden avulla haluttu toiminto suoritetaan vasta, kun käyttöliittymä on odotetussa tilassa. Nightwatch.js-työkalun valmiiden odotustoimintojen kanssa voidaan törmätä virhetilanteeseen, jos sivun sisällössä tapahtuu muutoksia käyttäjän syötteestä riippumattomissa ajankohdissa. Ongelman ratkaisemiseksi on mahdollista laatia mukautettu toiminto, jossa tämä otetaan huomioon. Nightwatch.js-työkalussa käytetään Selenium-kehystä [32]. Yhtenä mahdollisena ongelmana huomattiin, että testaaja ei voi helposti käsitellä Selenium-komentojen suorituksessa aiheutuneita virheitä, mikä voi vaikeuttaa yleisesti ja nopeasti toimivan komennon luomista.

Nightwatch.js luo oletuksena testiajoista koneellisesti luettavissa olevan testiraportin. Kehittäjälle saatavilla olevaan HTML-raportointiin tulee käyttää kolmannen osapuolen lisäosaa. Testitapauksien epäonnistuessa kyseisestä kohdasta voidaan ottaa kuvankaappauksia kytkemällä toiminto päälle testiajojen asetuksissa. Nightwatch.js-työkalun on saatavissa kolmannen osapuolen raportointilisäosia. Työkalun käytölle hyödyllisiä kolmannen osapuolen ohjelmistojen integraatioita ei ollut saatavilla työn kirjoitushetkellä.

Nightwatch.js-työkalun dokumentaatio koostuu sen verkkosivuilta löytyvistä ohjesivuista sekä ohjelmointirajapinnan dokumentaatiosta. Työkalun ohjelmakoodin säilytyspaikassa on saatavilla esimerkkejä sen käytöstä.

4.3.3 TestCafe

TestCafe-työkalusta käytettiin versiota 0.18.6. Työkalun käyttöön tarvitaan Node.js-ympäristö, jossa asennus voidaan tehdä npm-paketinhallintatyökalulla. Käyttäjän ei tarvitse asentaa erikseen selaimiin liittyviä ajureita.

TestCafe ei käytä sisäisesti Seleniumia, kuten muut vertailtavat työkalut [20]. Sen tarjoamat komennot sisältävät odotukset implisiittisesti, mikä helpottaa testitapauksien luomista. Testitapauksiin ei tarvinnut lisätä suoritettavaan toimintoon liit-

tymättömiä eksplisiittisiä odotuskomentoja ennen niiden suoritusta. Mikäli sivun sisällön muuttumista halutaan odottaa, se tehdään samaan tapaan niitä käyttäen kuten muissa vertailluissa työkaluissa. Ennen tapauksen suorituksen alkamista voidaan myös tarvita odotuskomento, jotta sivu ja sovelluksessa käytetty JavaScript-kirjastoa kerkeävät latautua.

Testitapauksia voidaan kirjoittaa JavaScript-skriptikielellä. TestCafella kirjoitettuja skriptejä voidaan laajentaa muilla JavaScript-kirjastoilla. Testiskriptien luonnissa voi käyttää page object -suunnittelumallia. TestCafe-työkalulle toteutetusta testitapauksesta löytyy koodilistauksesta 4.2. Testitapauksien toteuttamisen yhteydessä testaajan tulee antaa käsiteltävien elementtien tiedot ohjelmakoodissa. Testaaja voi hakea tiedot esimerkiksi selaimen kehittäjätyökaluilla.

```
import Page from './page';

const page = new Page();

fixture `My fixture`
    .page `https://devexpress.github.io/testcafe/example/`;

test('Text typing basics', async t => {
    await t
        .typeText(page.nameInput, 'Test')
        .expect(page.nameInput.value).eql('Test');
});
```

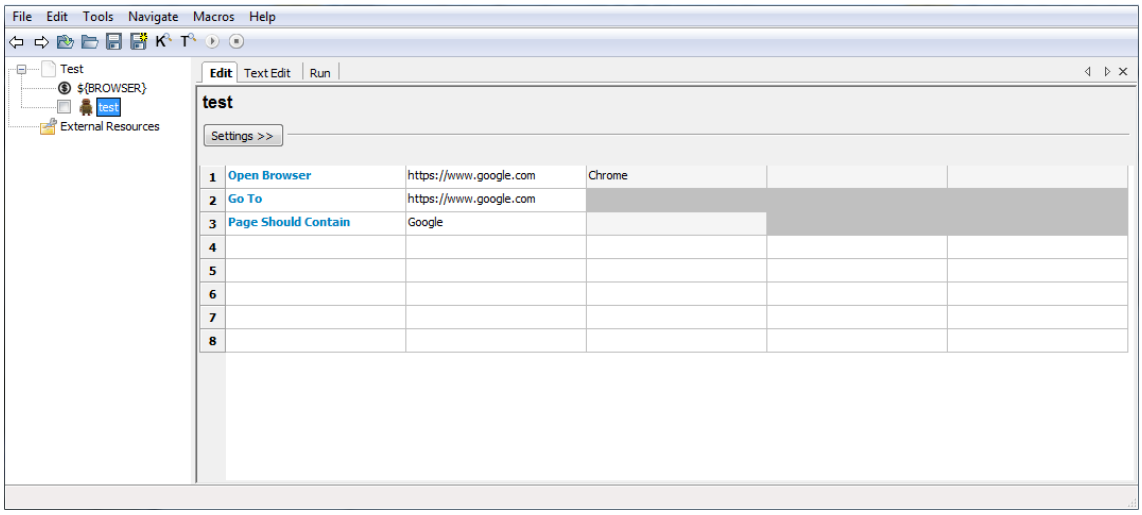
Ohjelma 4.2 *Esimerkki TestCafe-työkalulla toteutetusta testitapauksesta*

TestCafen dokumentaatio koostuu sen internet-sivuilta löytyvistä ohjesivuista ja ohjelmointirajapinnan dokumentaatiosta. TestCafe-työkalulle oli tarkasteluhetkellä saatavilla muutamia esimerkkejä sen lähdekoodin GitHub-tietolähteestä. TestCafen esimerkeissä käytetty tapa hyödyntää page object -suunnittelumallia on herkkä ylimääräiselle ylläpitotyölle, sillä niissä viitataan suoraan sivujen sisältämiin elementteihin sivujen yksityiskohtien piilottamisen sijaan.

TestCafe-työkalun raportointi tukee tekstimuotoisia ja koneellisesti luettavia raportteja, mutta tarkasteluhetkellä HTML-raportointia ei ollut saatavilla. TestCafe näyttää virhetapauksessa kohdan, jossa testitapauksen suoritus epäonnistui ja raporttiin on mahdollisuus sisällyttää selaimesta otettu kuvankaappaus virheen ajankohdasta.

4.3.4 Robot Framework

Robot Framework -työkalusta käytettiin versiota 3.0.2. Asennusta varten käytössä tulee olla Python-ympäristö. Robot Frameworkin kanssa on suositeltavaa käyttää Python 2.x-versioita, sillä sen IDE-työkalu tukee ainoastaan 2.x-versioita. Robot Framework vaatii selaimille ladattavan ajurin. Ajurin polku tulee lisätä path-muuttujaan. Testien nauhoittamiseen tarvitaan erillinen selainlisäosa.



Kuva 4.2 *Robot Framework -työkalun RIDE-käyttöliittymä*

Testitapausten kirjoittaminen tapahtuu käyttäen avainsanoja. Testitapauksiin luodaan sovelluksen käyttäjätoimintoja suorittavia avainsanoja. Robot Frameworkissa on mahdollista luoda uusia avainsanoja käyttäen jo olemassa olevia. Niitä varten voi luoda myös testikirjastoja Python-, Java- ja C-ohjelmointikielillä.

Testitapauksien kehittäjän tulee poimia testeissä käytettävät elementit ja syöttää niiden tiedot käyttämällä valitsimia. Tämän voi tehdä esimerkiksi tutkimalla sivun elementtejä web-selaimen kehittäjätyökalulla. Testitapauksen suoritettavissa komennoissa tulee huomioida asynkronisuus erikseen, sillä Robot Frameworkin komennot eivät sisällä implisiittisiä odotuksia. Asynkronisuus tulee huomioida testitapauksia laadittaessa, jotta testiajoista saadaan vakaita. Avainsanoihin pohjautuvissa testeissä ei tarvitse tuntea skriptikieltä testien toteuttamiseen. Asynkronisuuden erillisen huomioinnin ja mukautettujen avainsanojen luonnin tarpeen takia avainsanojen käyttö ei ole tässä yhteydessä merkittävästi yksinkertaisempaa kuin skriptien luonti.

Robot Frameworkin dokumentaatio koostuu sen verkkosivuilta löytyvistä ohjesivustoista sekä käyttäjäoppaasta. E2E-testaukseen on tarjolla SeleniumLibrary-kirjastolle saatavilla oleva dokumentaatio. SeleniumLibrary-kirjaston dokumentaatio koostuu sen sisältämien avainsanojen dokumentaatiosta. Sen mukana ei ollut kokonaisia esi-

merkkejä, mikä vaatii testaajan omaavan käsityksen Robot Frameworkin käytöstä ja osaamista hyödyntää avainsanadokumentaatiota kokonaisten testitapauksien luomiseen.

Robot Framework tuottaa raportteja koneellisesti luettavassa muodossa sekä HTML-muodossa. Raportteihin sisällytetään kuvankaappaus epäonnistuneiden testitapauksien päättymishetkestä. Tuotettuihin raportteihin sisältyy loki, josta saadaan tietoa testien suorituksen vaiheista ja suoritusajoista. Tämä helpottaa testitapauksen suorituksen tutkimista testiajon päättymisen jälkeen.

5. TULOKSET

Tässä luvussa kuvataan mihin valintoihin testaustyökalujen testaamisen jälkeen päädytään. Tämän lisäksi saatua tulosta ja sen mahdollisia käyttökohteita arvioidaan.

5.1 Pisteytysperusteet

Arvosana-asteikkona käytetään asteikkoa 0-5. Arvioinnissa otetaan myös huomioon kriteerin täyttymisen osalta se, miten hyvin niissä voidaan vähentää testiautomaatiossa ja päästä päähän -testauksessa ilmeneviä yleisiä ongelmia. Pisteytystä arvosteluasteikolle tarkennetaan löydettyjen puutteiden ja niiden keskenäisen vakavuusasteen mukaan.

Skriptituki: Ohjelmointikieli on yleisesti käytetty ja soveltuu tutkittavaan projektiin. Ohjelmointikielen käytön opettelu ei vaadi merkittävää määrää koulutusta eikä jaa testaajia sekä ohjelmoijia omiin ryhmiinsä. Ohjelmointikielessä voidaan hyödyntää valmista koodia.

Nauhoitus: Testitapausten osien nauhoitusta voidaan hyödyntää tapauksien luonnissa ja siitä saadaan hyötyä niiden ohjelmoinnin tueksi. Testitapausten osien nauhoituksesta saadaan hyötyjä verrattuna siihen, että niitä luotaisiin suoraan manuaalisten testitapausten pohjalta. Nauhoitus ei ole työkalun kannalta pakollinen, mutta toimiessaan se on hyödyllinen. Kriteerille annetaan painoarvo 0,5.

Raportointi: Raportit saa tallennettua koneellisesti luettavassa muodossa sekä ihmisille luettavassa muodossa. Työkalun tuottamat raportit tuottavat tietoa, joka on hyödyllistä testaajille. Integraatiot muihin järjestelmiin ovat kriteerin kannalta positiivisia.

Dokumentaatio: Dokumentaatio on toimiva, ajantasainen ja selkeä. Se auttaa työkalun käytössä ja vähentää tarvittavaa koulutuksen määrää ja mahdollistaa itsenäisen työskentelyn. Dokumentaation monipuolisuus ja käyttöesimerkit parantavat sen hyödyllisyyttä.

Helppokäyttöisyys: Työkalu on helppo ottaa käyttöön ja sen käyttäminen on suju-

vaa. Se tuottaa testiajoista hyödyllistä tietoa testitapauksen suorituksesta. Helppokäyttöisyys on subjektiivinen kriteeri ja on arvioitu kehittäjän näkökulmasta. Siinä arvioidaan muun muassa työkalun asennusta, testitapauksien luomista ja niiden ajamista. Helppokäyttöisyys on merkittävä kriteeri ja sille annetaan korkea painoarvo 2.

Ylläpidettävyys: Työkalulla luotuja testitapauksia on helppo ylläpitää, kun niihin tulee muutostarpeita. Ylläpidettävyys on tärkeää testauksen kustannustehokkuuden kannalta ja sille annetaan muita kriteereitä korkeampi painoarvo 3.

Vakaus: Työkalulla hyvin luodut testitapaukset läpäisevät oikein toimivan ohjelman testiajot vakaasti. Kriteeriä arvioidaan sen mukaan, miten vakaasti testiajot suorittuvat läpi. Tämän kriteerin odotetaan täyttyvän kaikissa työkaluissa niin, että testiajot saadaan toimimaan vakaasti, mikäli testitapaukset on luotu oikein. Epäluotettavasti toimiva työkalu on suuri ongelma testityökalulle, jonka takia kriteerille annetaan tavallista korkeampi painoarvo 2. Ongelmien ilmetessä testiajoja suoritettaessa ensin varmistetaan, että testitapaus on rakennettu oikein, ja että ongelma ei johdu testityökalun ulkopuolisesta syystä.

Tuotetestauksen tulokset vaikuttavat pisteytyksessä dokumentaation, helppokäyttöisyyden, ylläpidettävyyden ja vakauden arviointiin. Aktiivisuutta ja käytön tukea ei arvioitu tässä yhteydessä enää erikseen, sillä niiden suhteen on tehty karsinta aiemmassa vaiheessa. Tuotteiden sijoittaminen arvosteluasteikolle on näiden kriteerien kohdalla vaikeaa. Käytön tukea voidaan arvioida paremmin pitkän aikavälin kokemuksien avulla.

5.2 Pisteytys

Työkalujen pisteytys IoT-Ticket -ohjelmistotuotteelle on esitetty taulukossa 5.1. Kriteerien painoarvo on esitetty niiden nimen yhteydessä sulkeissa.

Työkalujen pisteytys EcoReaction-ohjelmistotuotteelle on esitetty taulukossa 5.2.

Skriptituen osalta kaikissa työkaluissa oli saatavilla skriptikieli, jota käytetään tutkittavissa ohjelmistoprojekteissa vähintään yhdellä osa-alueella. Laajennettavuus oli yksinkertaista Nightwatch.js-, TestCafe- ja Robot Framework -työkaluissa, joissa laajennuksia voidaan ottaa käyttöön riippuvuudenhallintatyökalulla. Katalon-työkalussa laajennukset tulee liittää mukaan JAR-paketteina.

Vertailtavista työkaluista Katalon- sekä Robot Framework -työkaluihin oli saatavilla

	Katalon	Nightwatch.js	Robot Framework	TestCafe
Skriptituki (1)	4	5	5	5
Nauhoitus (0,5)	1,5	0	0	0
Raportointi (1)	4	4	4	4
Dokumentaatio (1)	4	4	3	4
Helppokäyttöisyys (2)	6	4	4	8
Ylläpidettävyys (3)	12	12	12	15
Vakaus (2)	10	10	10	10
Yhteensä	41,5	39	38	46

Taulukko 5.1 *Testityökalujen pisteytys IoT-Ticketille*

	Katalon	Nightwatch.js	Robot Framework	TestCafe
Skriptituki (1)	4	5	5	5
Nauhoitus (0,5)	1,5	0	0	0
Raportointi (1)	4	4	4	4
Dokumentaatio (1)	4	4	3	4
Helppokäyttöisyys (2)	6	4	4	8
Ylläpidettävyys (3)	12	12	12	15
Vakaus (2)	10	10	10	10
Yhteensä	41,5	39	38	46

Taulukko 5.2 *Testityökalujen pisteytys Ecoreaction-tuotteelle*

nauhoitustyökalu. Katalonin nauhoitustuki sisältyi työkaluun ja Robot Frameworkiin nauhoitus on saatavissa kolmannen osapuolen liitännäisen kautta. Katalonin nauhoitustuki voi auttaa testitapauksien luonnissa, mutta se tulee tietyin rajoituksin. Robot Frameworkille kokeiltiin kolmannen osapuolen Robotcorder-lisäosaa. Sen tuottamat skriptit eivät huomioineet testitapauksien suoritukseen sisältävää asynkronisuutta oikein eikä elementtien poiminta toiminut luotettavasti. Muita vaihtoehtoja ei löydetty kartoitukseen tehdyssä haussa. Yhtenä nauhoitustyökalujen hyötynä nähtiin mahdollisuus käyttää niitä testitapauksissa, joissa tarkastellaan asiakassisällön oikeellisuutta. Nauhoitustyökalujen heikkouksien takia nauhoitusta ei voida kuitenkaan hyödyntää halutulla tavalla.

Kaikki tutkitut työkalut tukivat selaintueltaan testien ajamista Chrome-, Firefox-, Internet Explorer- ja Edge-selaimilla. Testien laatimisessa Nightwatch.js- ja Robot Framework -työkaluilla huomattiin eroja komentojen suorituksessa selaimien välillä, mutta ongelmat saatiin ratkaistua luomalla uusi komento yhteensopivuuden korjaamiseksi. Selaintuen toimivuutta kaikilla vaadituilla selaimilla ei kokeiltu, minkä takia selaintuki jätettiin pois päätöksentekokriteereistä.

Raportointiominaisuudet todettiin kaikissa työkaluissa riittäviksi kehityksen osalta.

Niissä kaikissa saatiin tieto testitapauksien onnistumisesta ja kestosta. Virhetapauksissa saatiin testitapauksen suorituspolku sekä kuvankaappaus virhekohdasta. Testitulokset voitiin tallentaa kaikissa työkaluissa myös koneellisesti luettavassa XML-formaatissa, jota voidaan käyttää integraatioissa muihin järjestelmiin, kuten testitulosanalyysityökaluihin. Katalon on vertailluista työkaluista ainut, jossa on mukana valmiita integraatioita projektinhallinta- ja testitapaustenhallintatyökalujen kanssa.

Käyttöjärjestelmien osalta suurin osa vertailuun valituista työkaluista tukee yleisimpiä Windows-, Linux- ja MacOS-käyttöjärjestelmiä. Katalon-työkalun käyttöliittymä ei ollut saatavissa Linux-käyttöjärjestelmälle. Tämä puute oli merkittävä, sillä uusia testitapauksia on vaikea luoda Katalon-työkalussa ilman sen graafista käyttöliittymää.

Dokumentaatio oli riittävä kaikissa testatuissa työkaluissa, jotta niillä voitiin luoda testitapauksia itsenäisesti. Tärkeimmäksi dokumentaatioksi tässä työssä osoittautui suoritetussa kokeilussa ohjelmointirajapintojen dokumentaatio sekä työkalun käyttöä havainnollistavat esimerkit. Jälkimmäiset helpottivat testauksessa merkittävästi työkalujen käyttöönottoa. Katalon-työkalun dokumentaatio oli muita työkaluja monipuolisempi, sillä siihen sisältyi kokonaisia tutoriaaleja sekä ohjevideoita. Tavallista erikoisemmissa testitapauksissa toteutustekniikoista saatiin tietoa työkalujen yhteisön keskustelukanavien kautta.

Testauksessa huomatut testien vakauteen liittyvät ongelmat liittyivät siihen, miten testitapauksien kirjoittajan tulee huomioida niiden suorituksen asynkronisuus. Kaikissa muissa valituista työkaluista paitsi TestCafe-työkalussa testitapauksen kirjoittajan pitää erikseen huomioida tapaukset, joissa haetut elementit eivät ole suoraan näkyvissä tai niille ei voi suorittaa haluttuja toimintoja heti. Kaikkiin näistä muista työkaluista on mahdollista toteuttaa toimintoja, jotka huomioivat asynkronisuuden, mutta ne monimutkaistavat testiautomaation toteutusta. Työkaluissa, joissa komennot sisältävät tarvittavat implisiittiset odotukset, testiautomaation rakentaminen on helpompaa.

Työkaluilla saatiin luotua testitapauksia, jotka suoriutuivat vakaasti läpi peräkkäisillä suorituskerroilla. Tämän perusteella testitapauksien vakauteen tulee kiinnittää huomiota niitä laadittaessa. Toimivien testitapauksien luomiseksi tarkastelua vaativat kohdat voidaan tunnistaa käyttöliittymää tutkimalla. Testitapauksien vakauden arvioinnissa käytettyä menetelmää ajaa ne useampaan kertaan ei voida pitää luotettavana merkkinä testitapauksen oikeellisuudesta, sillä testiajossa esiintyvät viiveet voivat vaihdella riippuen testien ajoympäristöstä. Vakauden arvioinnissa ei päästy näkemään pidemmän aikavälin käytön tuloksia, eikä testitapauksien ajoa tehty pal-

velinympäristössä. Eroina paikalliseen ympäristöön verrattuna ovat muun muassa potentiaaliset uudet virhekohdat palvelimien välisessä kommunikaatiossa, vikasietoisuudessa sekä mahdollisesti lisääntyvissä viiveissä. Niissä saatetaan tarvita myös ylimääräisiä ohjelmistoasennuksia selaimia varten, jotka eivät tue headless-tilaa. Selenium-pohjaisissa työkaluissa jatkuvassa integraatiossa voidaan ottaa käyttöön etäpalvelimella ajossa oleva Selenium-palvelin.

Testauksessa törmättiin kahteen erilaiseen virhetilanteeseen, joista toisessa sivuston sisältö muuttui ennustamattomassa ajassa. Tämä aiheutti Katalon-, Nightwatch.js- ja Robot Framework -työkaluissa satunnaisesti Stale Element -virheen. Ongelma voitiin kiertää kirjoittamalla oma toiminto elementin ilmentymisen tai katoamisen odottamiseen, jolla virhetilanteelta vältyttiin. Ongelma voidaan ratkaista myös yksinkertaisemmin lisäämällä vakiopituinen odotus, mutta sen haittapuolena on testitapauksen suorituksen hidastuminen ja toiminnan epävarmuus. Tapauksissa, joissa tarkasteltava elementti muuttuu useampaan kertaan sen käsittelyn yhteydessä, tarvitaan mukautettu toiminto. Omat komennot voivat aiheuttaa ylimääräistä ylläpitotyötä, mikäli testaustyökalun tarjoama rajapinta muuttuu. Toinen ongelmista liittyi elementtien klikkaukseen. Mikäli klikattava elementti oli toisen elementin alla, testitapauksen suoritus päättyi virheeseen. Tämä ongelma voidaan ratkaista lisäämällä toiminto, jolla odotetaan, että peittävä elementti katoaa.

Helppokäyttöisyyttä voitiin arvioida tutkimalla työkalun käytön ja testitapauksien toteutuksen monimutkaisuutta. Testitapauksien toteutettavuutta ja vaadittua työmäärää voitiin tutkia selvittämällä liittyikö niihin käyttöliittymätoimintoja, jotka ovat toteuttavissa testaustyökalun tarjoaman rajapinnan kautta. Mikäli työkalu ei tarjoa toiminnon suorittamiseen rajapintaa, sitä varten tulee luoda mukautettu toiminto. Mikäli testityökaluun integroiduilla toiminnoilla ei saada toteutettua vakaita testitapauksia, voidaan myös joutua luomaan uusi mukautettu toiminto.

Selenium-pohjaisissa työkaluissa tarvittu eksplisiittisten odotuksien erillinen käsittely hidasti testitapauksien luomista, sillä synkronointiin liittyviin ongelmiin voidaan törmätä testitapausta kokeiltaessa. Elementtejä joudutaan tutkimaan käsin, jonka jälkeen testitapauksiin lisätään vaaditut odotukset. Eksplisiittisen odotuksen suorittavia komentoja ei myöskään usein saada lisättyä tai korjattua yhdellä kerralla, sillä testitapauksen suoritus lakkaa usein, kun suorituksessa kohdataan yksi virhe. Ne vaikeuttavat myös testitapauksien ylläpitoa, sillä niihin voidaan joutua tekemään muutoksia käyttöliittymän muuttuessa. Eksplisiittiset odotukset tuli huomioida erikseen Katalon-, Nightwatch.js- ja Robot Framework -työkaluissa. TestCafe-työkalulla luoduissa testitapauksissa tarvitsi huomioida vähemmän yksityiskohtia kuin muilla vertailluilla työkaluilla. TestCafe sisältää myös odotuskomentoja, joita voidaan hyö-

dyntää kun web-sivun sisällön muuttumista tulee odottaa tai kun sisäänrakennetun implisiittisten odotuksien kanssa törmätään ongelmiin.

Testitapauksien laatimista voidaan yksinkertaistaa, mikäli tilan muutokset saadaan indikoitua testattavassa ohjelmassa riittävän nopeasti. Tällöin voi olla mahdollista käyttää mukautettua käyttöliittymäkomentoa, joka sisältää tarvitut eksplisiittiset odotukset, vähentäen tarvetta lisätä niitä testitapauksien ohjelmakoodiin. Tarvittavien eksplisiittisten odotusten suorittavan funktion toteutus voi olla joko ohjelmaspesifinen tai yleiskäyttöinen. Ohjelmaspesifisen odotusfunktion luominen on mahdollisesti verrattaen helppoa, mikäli tilamuutoksien indikoimiseen käytetään aina samalla tavalla tunnistettavaa mekanismia. Yleiskäyttöisen odotusfunktion pitää pystyä havaitsemaan sovelluksen tila niin, että käytetyssä testaustyökalussa ei törmätä ongelmiin. Odotusfunktio voi olla käytännössä vaikea saada toimimaan halutulla tavalla kaikkien sovelluksien kohdalla. Odotuskomennon käyttö voi hidastaa testitapauksien suoritusnopeutta, mikäli sitä käytetään paljon kohdissa, joissa eksplisiittisiä odotuksia ei tarvita. Eksplisiittisten odotusten yhtenäistäminen on tällöin toinen vaihtoehto.

Katalon- ja Robot Framework -työkalujen testitapauksia varten oli mahdollista tunnistaa elementtejä käyttöliittymän avulla. Katalonin käyttöliittymä nopeutti elementtien tunnistamista, mutta ei poistanut tarvetta elementtien paikantamistavan manuaaliseen etsimiseen ja valintaan kokonaan. Toiminnallisuus oli saatavissa Robot Frameworkille vain kolmannen osapuolen lisäosan kautta, eikä siinä saatu yhtä paljon hyötyjä testitapauksien luomiseen kuin Katalon-työkalussa.

Ylläpidettävyyttä parantavina ominaisuuksina kaikissa kokeilluissa työkaluissa voitiin käyttää hyödyksi abstraktioita ja uudelleenkäyttöä. Aikasynkroinnin helppous vaikuttaa myös testien ylläpidettävyyteen. TestCafe-työkalulla luotuja testitapauksia on tältä osin helpompi muokata, sillä siinä synkronointiin ei usein tarvitse kiinnittää yhtä paljon huomiota testitapauksia luodessa.

Suorituskykymittauksien tuloksissa nähtiin, että testitapauksien suorituskyky oli samankaltainen kaikissa työkaluissa sekä IoT-Ticket- että EcoReaction-tuotteissa. Suorituskyvyn parantamiseksi Katalon-, Nightwatch.js- ja Robot Framework -työkaluja käytettäessä ohjelman tilan näkyvyyttä voidaan parantaa niin, että testiajossa pystytään havaitsemaan nopeammin milloin komentojen suoritusta voidaan jatkaa. Näissä työkaluissa tulee kiinnittää enemmän huomiota testattavuuteen kehityksen aikana, jotta ohjelman toiminnallisuutta ei tarvitse muokata myöhemmin sen parantamiseksi.

Tutkittujen työkalujen soveltuvuus käytettäväksi tutkituissa ohjelmistotuotteissa

riippuu testauksen tavoitteista. Testauksen perusteella työkaluja voidaan käyttää platform-ominaisuuksien testaukseen. Testaustyökalut eivät olleet riittäviä asiakassisällön testaukseen, jota tehtäisiin platform-testauksen ulkopuolella käyttäen nauhoitustyökaluja. Suurimpina ongelmina on nauhoitustyökaluilla tuotettujen testitapauksien heikko ylläpidettävyys ja tuotettujen testitapauksien vakaus. Mikäli nauhoitustyökalut tuottaisivat vakaasti toimivia testitapauksia, niitä tulisi kuitenkin nauhoittaa uudelleen, kun käyttöliittymään tai ohjelman logiikkaan tulee muutoksia.

Yksi vaihtoehto asiakassisällön testauksen parantamiseksi on tunnistaa testattavista toiminnoista tuotteen platform-ominaisuudet ja testata niitä käyttäen tunnettuja syötteitä ja ulostuloja. Platform-ominaisuuksilla tarkoitetaan tässä toiminnallisuutta, joka voidaan tunnistaa ohjelmistotuotteen yleisenä toiminnallisuutena, eikä ole asiakasspesifistä. Tämänlaisella testaamisella varmistettaisiin, että asiakasspesifisen sisällön muodostavat toiminnallisuudet säilyvät ehjinä, kun niihin tehdään muutoksia ohjelmistotuotteessa. Asiakassisällön oikeellisuus tarkistettaisiin tällöin erillään ohjelmistotuotteen omasta testauksesta manuaalisesti. Ohjelmistotuotteessa voidaan myös mahdollisesti tarjota käyttäjälle työkaluja, joilla asiakassisällön oikeellisuutta voi testata ohjelmistotuotteen avulla. Tämän lähestymistavan haittapuolena on siitä aiheutuva ylimääräinen toteutustyö. Tässä tapauksessa myös tarjottu testausrajapinta tulee huomioida tuotteen platform-ominaisuuksien testauksessa.

Käyttöliittymätestitapauksissa on paljon ylläpitotyötä ja niiden suoritus on hidasta. Käyttötapauksien testaamisessa tulee ensin tarkastella mistä osista tapaus koostuu ja missä kohdassa järjestelmää ne voidaan testata. Mikäli käyttötapaus voidaan testata ilman käyttöliittymää, on hyvä tarkastella voidaanko sen koostamia osia testata yksikkö- ja integraatiotestauksen avulla. Tällä tavoin toiminnallisuutta saadaan testattua eristetysti järjestelmän muista osista, jolloin ongelmien syitä voidaan tutkia helpommin. Testitapauksista saadaan tällöin myös suoritusajaltaan lyhyempiä ja niistä saadaan nopeampaa palautetta. E2E-testausta voidaan käyttää silloin, kun järjestelmää halutaan testata kokonaisuudessaan. E2E-testauksessa voidaan keskittyä kokonaisiin käyttötapauksiin. Siinä ei ole suositeltavaa testata samoja käyttötapauksia suurella määrällä erilaisia syötteitä, koska tällöin testiajojen suorituspituudet kasvavat nopeasti suuriksi.

Tulosten perusteella käyttöönottoon suositeltaisiin TestCafe-työkalua. Se osoittautui tehdyssä testauksessa helppokäyttöiseksi, koska sen käytössä tarvittiin vähemmän lisättyjä eksplisiittisiä odotuskomentoja. Katalon-työkalu sisälsi nauhoitusominaisuuksia, joita TestCafe-työkalu ei sisältänyt. Helppokäyttöisyys ja testien ylläpidettävyys todettiin tärkeämmiksi, jonka takia Katalon jäi pisteytyksessä toiseksi.

Pisteytykset olivat samat IoT-Ticket- ja Ecoreaction -tuotteille, mikä toteutui ohjelmistotuotteissa olleiden samankaltaisuuksien takia. Testityökalua käyttävien testaajien oletetaan osaavan ohjelmointia. Kaikissa kokeilluissa työkaluissa vaaditaan ohjelmointitaitoja hyvin ylläpidettävän testiautomaation rakentamiseksi.

6. ARVIOINTI

Työn suorituksessa löydettiin toimiva prosessi testaustyökalun valintaan. Karsintavaiheessa työkalujen määrää saatiin vähennettyä merkittävästi tarkastelemalla työkaluja pintapuolisella tasolla. Tutkimusosuuden tarkemmassa tutkinnassa karsintaan ja päätökseentekoon saatiin muodostettua systemaattinen prosessi. Työssä suoritetun tutkimuksen lähtökohdat osoittautuivat myös järkeviksi, sillä tutkitut ohjelmistotuotteet olivat testiautomaation kannalta riittävän samankaltaisia, jotta niille voitiin suunnitella yhdessä soveltuvaa testiautomaatioratkaisua.

Karsinnan kriteerit toimivat hyvänä tapana vähentää iso osa saatavilla vaihtoehdoista pois käyttämättä niiden tarkasteluun merkittävää määrää aikaa. Karsinnan jatkaminen voi osoittautua huonoksi silloin, kun vaatimuksia lisätään ainoastaan työkalujen määrän vähentämiseksi. Mahdollisten lisättävien kriteerien keskimääräistä tärkeysjärjestystä tulisi arvioida subjektiivisuuden vähentämiseksi. Osaa testaustyökalujen karsinnassa käytetyistä kriteereistä ei saada jäljitettyä lähteeseen tai niissä käytettiin arvioijan omaa subjektiivista harkintaa. Kriteereitä ei voitu myöskään muodostaa kaikilta osin, sillä valintaprosessista puuttuivat eri sidosryhmät. Käyttöaktiivisuus osoittautui epävarmaksi kriteeriksi joidenkin kaupallisten tuotteiden kohdalla, sillä niissä ei näkynyt julkista käyttöaktiivisuutta, mutta asiakasreferenssien mukaan niillä kuitenkin on käyttäjiä.

Tiedonhaussa löydettiin Google-hakukoneen avulla suorien hakutuloksien lisäksi työkaluja viidestä testityökalulistauksesta. Tämä paransi tiedonhaun lähteiden monipuolisuutta, mutta niitä olisi voinut olla enemmän. Monet löydetyistä työkaluista esiintyivät vain yhdessä tiedonhakulähteessä. Mahdollisia parannuskohteita tiedonhakuun ovat muun muassa hakukoneen käyttö kattavammalla määrällä hakusanoja sekä haku suoraan sivustoilta, joilta voidaan löytää ohjelmistotuotteita.

Työkalujen haussa monien tarkistettavien vaihtoehtojen voidaan olettaa jääneen löytymättä. Tämän syynä ovat puutteet tiedonhakuun käytetyissä lähteiden monipuolisuudessa, hakutermeissä sekä hakuun käytetyssä rajallisessa ajassa. Haussa olisi voitu hyötyä työkalujen olemassa olevien käyttäjien kokemuksista. Tällä tavoin työkalusta oltaisiin mahdollisesti voitu saada tietoa, mikä tulisi muutoin ilmi vasta,

kun sitä on käytetty merkittävä aika tai sitä varten toteutettuun testiautomaatioon on tehty merkittävästi työtä.

Vertailuun valituilla testitapauksilla voitiin tutkia monia E2E-testiautomaatiossa ilmeneviä yksityiskohtia ja ongelmia. Testityökalun valinnassa olisi tullut käyttää tukena E2E-testaussuunnitelmaa sen sijaan, että valinnan tukena käytettiin erikseen laadittuja testitapauksia. Tällä ei kuitenkaan ollut merkittävää vaikutusta suoritettuun vertailuun, sillä luotujen testitapauksien määrä oli pieni.

Tarkempaan vertailuun otettuja testaustyökaluja käytettiin melko vähän aikaa, minkä takia työkalujen pidempiaikaisesta toimivuudesta ei saatu varmuutta. Testaus suoritettiin paikallisessa ympäristössä lyhyellä aikavälillä, minkä takia testiajojen vakauteen liittyviä ongelmia voi mahdollisesti ilmetä, kun testiajoja on suoritettu enemmän kuin työssä suoritetussa testauksessa. Kokeilussa ei päästy myöskään kokeilemaan työkalun skaalautuvuutta isoihin määriin testitapauksia. Työkaluja kokeiltaessa suoritetuista testiajoista saatiin suorituskykytietoa, mutta tuloksista ei voitu päätellä kuinka paljon suoritusajat eroavat testitapauksien määrän ollessa suuri. Laajojen testiajojen kestoa ja työkalujen välisiä eroavaisuuksia oltaisiin myös voitu analysoida helpommin, mikäli testiajoja olisi kokeiltu laajemmalla määrällä testitapauksia. Toinen merkittävä pidemmällä aikavälillä arvioitava asia on työkalun päivittyvyys ja toimivuus eri selaimien kanssa. Arvioinnissa olisi voitu yrittää hyödyntää enemmän muiden käyttäjien kokemuksia pidempiaikaisesta käytöstä.

Työkalujen käyttöjärjestelmätuen tarkempi tutkiminen voidaan toteuttaa valinnan ensisijaisten kriteerien täytyttyä. Tutkimuksessa ei kokeiltu työkalujen toimivuutta muilla käyttöjärjestelmillä, mutta kokeilu on syytä tehdä, mikäli tukea muille käyttöjärjestelmille tullaan tarvitsemaan. Mobiililaitetestausta tukevien työkalujen toimivuus mobiilitestauksessa on hyvä kartoittaa tarkastelun yhteydessä. Tällä tavalla voidaan selvittää etukäteen, onko tarvetta lähteä erikseen etsimään työkalua mobiililaitteiden testaukseen. Testityökalun sopivuutta voidaan samalla tutkia muiden testauksen tyyppien, kuten visuaalisen testauksen osalta.

Vertailtujen työkalujen toimivuutta ei kokeiltu jatkuvan integraation järjestelmissä. Tämän tutkiminen on tärkeää ennen kuin työkalu otetaan merkittävissä määrin käyttöön. Valintaprosessissa tämä vaihe sijoitettaisiin työkalun kokeilun jälkeen. Kokeilu voi tuoda arviointiin mukaan uusia tarkastelukohtia muun muassa helppokäyttöisyyden, vakauden ja dokumentaation osalta. Merkittävien ongelmien ilmetessä siirryttäisiin tarkastelemaan muita vaihtoehtoja. Valintaprosessissa tulisi myös suunnitella etukäteen missä vaiheissa käyttöönottoa lähdetään laajentamaan kokeilun jälkeen.

Työkalujen vertailuissa saatiin selvitettyä niiden välisiä eroja käytetyissä kriteereissä. Ratkaisevat erot huomattiin testityökalujen sisältämissä ominaisuuksissa ja tämä helpotti niiden keskinäistä vertailua ja pisteytystä. Arviointiin käytetyn painotetun summan menetelmässä painoarvojen valinta oli subjektiivista. Arvioinnissa ei arvioitu testityökalujen hintaa suoraan käyttäen kustannuslaskentaa, mikä oli perusteltua toteutetun valintaprosessin koon pohjalta. Hintojen arvioinnin avulla työkalujen kustannuksiin olisi voitu ottaa kantaa kartoitusvaiheen jälkeen tehdyssä karsinnassa.

Työkalujen arviointi tehtiin testiautomaatio-osaajan näkökulmasta. Helppokäyttöisyysden ja dokumentaation subjektiivinen arviointi vain yhtä näkökulmaa hyödyntäen voi aiheuttaa myöhemmin ongelmia, kun testityökalu otetaan käyttöön laajemmin. Testityökalun valinnassa olisi tullut olla mukana useita eri henkilöitä, jotta valinnassa olisi tullut ohjelmistoprojektien eri osapuolien näkemyksiä.

7. YHTEENVETO

Työn kirjallisuusosassa tutkittiin päästä päähän -testausta ja työkalujen valintaprosesseja. Näiden avulla muodostettiin valintaprosessi, jossa otettiin huomioon E2E-testaustyökalujen ominaisuudet. Työn tutkimusosuudessa suoritettiin valintaprosessi testaustyökalun valintaan. Hakukoneella löydettiin riittävä määrä lähteitä, jonka jälkeen työkaluja karsittiin käyttäen kirjallisuuden avulla selvitettyjä yleisiä valintakriteerejä. Yleisten kriteerien avulla työkalujen määrä saatiin vähennettyä alle puoleen alkuperäisestä määrästä. Nämä kriteerit todettiin hyödylliseksi tavaksi vähentää tutkittavien vaihtoehtojen määrää tarvitsematta tutkia vaihtoehtoja liian yksityiskohtaisesti. Tällä tavoin vaihtoehtojen tutkimiseen käytettyä aikaa voidaan saada vähennettyä merkittävästi.

Tämän jälkeen vaihtoehtojen määrää vähennettiin tiukentamalla kriteerejä ja käyttämällä apuna valinnan käyttökohteille muodostettuja kriteerejä. Työkalujen määrä saatiin vähennettyä edelleen riittävän pieneksi, jotta niitä voitiin tutkia tarkemmin. Vertailuun valittiin työkaluiksi Katalon, Nightwatch.js, TestCafe ja Robot Framework.

Työkaluille suoritettiin vertailu, jossa testaustyökaluille luotiin testitapauksia IoT-Ticket- ja EcoReaction-tuotteille. Nämä tapaukset saatiin luotua molemmille ohjelmistotuotteille onnistuneesti. Tehdyn vertailun perusteella suositelluksi testityökaluksi jatkoarviointiin valittiin TestCafe, jonka ratkaiseviksi eduiksi todettiin helppokäyttöisyys sekä testitapauksien hyvä ylläpidettävyys. Hyvä ylläpidettävyys on tärkeä ominaisuus, sillä loppukäyttöä kuvaavat käyttöliittymätestitapaukset voivat muuttua usein.

Tietoja päästä päähän -testauksessa ilmenevissä yleisistä ongelmista voidaan hyödyntää valintakriteerien muodostuksessa sekä työkaluna tunnistaa kriteerien täyttyvyyttä tehtäessä vertailua. Työssä toteutettua prosessia on mahdollista käyttää tukena työkaluvalintaprosessin muodostamisessa ja toteutuksessa. Sitä tulee muokata soveltumaan tehtävään työkaluvalintaan. Valintaa tehdessä työkalun sopivuutta tulee tutkia testauksen kokonaisratkaisussa. Tämän jälkeen sen käyttöönottoa voidaan laajentaa.

Tuloksien avulla voitiin todeta, että ohjelmistotuoteprojekteissa on mahdollista hyödyntää samoja testaustyökaluja, mikäli projektit sisältävät riittävästi samankaltaisuuksia. Valinnan toteuttamiskelpoisuutta tulee arvioida ennen kuin sitä lähdetään toteuttamaan.

LÄHTEET

[1] End-to-End Test. https://www.techopedia.com/definition/7035/end-to-end-test. Haettu 7.11.2017

[2] Myers, J., Sandler, C. 2004. The Art of Software Testing. John Wiley & Sons. 234 p.

[3] Humble, J., Farley, D. 2011. Continuous Delivery : Reliable Software Releases Through Build, Test, and Deployment Automation. Pearson Education. 463p.

[4] Chen, L. 2015. Continuous Delivery: Huge Benefits, but Challenges Too. IEEE Software, Vol 32(2), pp. 50-54.

[5] Fewster, M., Graham D. 1999. Software Test Automation: Effective Use of Test Execution Tools. ACM Press/Addison-Wesley Publ. Co., New York, NY, USA. 574p.

[6] Kaner, C., Bach, J., Pettichord, B. 2002. Lessons Learned in Software Testing: a context-driven approach. John Wiley & Sons, Inc., New York, NY, USA. 286p.

[7] Schieferdecker, I. 2012. Model-Based Testing. IEEE Software, Vol. 29(1). pp. 14-18

[8] PageObject. https://martinfowler.com/bliki/PageObject.html. Haettu 27.2.2018

[9] Wayne Matthias Roseberry. 2016. Winning with Flaky Test Automation. `http://uploads.pnsqc.org/2016/papers/12.WinningWithFlakyTestAutomation.pdf`

[10] IoT-Ticket Platform. https://iot-ticket.com/platform. Haettu 31.3.2018

[11] Ecoreaction. https://www.wapice.com/fi/tuotteet/ecoreaction. Haettu 31.3.2018

[12] IEEE Recommended Practice for Software Acquisition, IEEE Std 1062-2015 (Revision of IEEE Std 1062-1993), pp.1-69, Feb. 26 2016

[13] Roy, B. The Optimisation Problem Formulation: Criticism and Overstepping. The Journal of the Operational Research Society, Vol. 32, No. 6 (Jun., 1981), pp.427-436

[14] IEEE Standard for Adoption of ISO/IEC 14102:2008 Information Technology–Guideline for the Evaluation and Selection of CASE Tools. 2010. IEEE Std 14102-2010, pp. 1-59

[15] Ishizaka, A., Nemery, M. Multi-Criteria Decision Analysis : Methods and Software. John Wiley & Sons. 296p.

[16] Belton, V., Stewart, T. 2002. Multiple Criteria Decision Analysis: An Integrated Approach. Springer US. 372p.

[17] Roy, B. 2013. Multicriteria Methodology for Decision Aiding, Springer US. 293p.

[18] Quality management systems. Suomen Standardoimisliitto, SFS-EN ISO 9001, Helsinki, 2008.

[19] Stale Element Reference Exception. https://www.seleniumhq.org/exceptions/stale_element_reference.jsp. Haettu 24.4.2018

[20] TestCafe FAQ. `http://devexpress.github.io/testcafe/faq/`. Haettu 6.5.2018

[21] QF-Test. `https://www.qfs.de`. Haettu 6.5.2018

[22] Ranorex. `https://www.ranorex.com/`. Haettu 6.5.2018

[23] Sahi Pro. `http://sahipro.com/`. Haettu 6.5.2018

[24] TestComplete. `https://smartbear.com/product/testcomplete/overview/`. Haettu 6.5.2018

[25] Telerik Test Studio. https://www.telerik.com/teststudio. Haettu 6.5.2018

[26] Watir. `http://watir.com/`. Haettu 6.5.2018

[27] Katalon Studio. `https://www.katalon.com/`. Haettu 30.4.2018

[28] TestCafe. `https://devexpress.github.io/testcafe/`. Haettu 30.4.2018

[29] Serenity. `http://www.thucydides.info`. Haettu 6.5.2018

[30] Robot Framework. `http://robotframework.org/`. Haettu 30.4.2018

[31] Squish. `https://www.froglogic.com/squish/`. Haettu 6.5.2018

[32] Nighwatch.js. `http://nightwatchjs.org/`. Haettu 30.4.2018

[33] Protractor. `https://www.protractortest.org`. Haettu 6.5.2018

[34] Eggplant functional. https://eggplant.io/. Haettu 6.5.2018

[35] iMacros. https://imacros.net/. Haettu 6.5.2018

[36] Sahi. http://sahipro.com/sahi-open-source/. Haettu 6.5.2018

[37] Selenium IDE. https://www.seleniumhq.org/projects/ide/. Haettu 6.5.2018

[38] Trincentis Tosca. https://www.tricentis.com/software-testing-tools/. Haettu 6.5.2018

[39] Unified Functional Testing. https://software.microfocus.com/en-us/products/unified-functional-automated-testing/overview. Haettu 6.5.2018

[40] TestingWhiz. https://www.testing-whiz.com/. Haettu 6.5.2018

[41] ZapTest. https://www.zaptest.com/. Haettu 6.5.2018

[42] Helium. https://heliumhq.com/. Haettu 6.5.2018

[43] Canoo WebTest. http://webtest.canoo.com/. Haettu 6.5.2018

[44] Buster.js. http://busterjs.org/. Haettu 6.5.2018

[45] Web Test. http://www.appperfect.com/products/web-test.php. Haettu 6.5.2018

[46] RedwoodHQ. http://redwoodhq.com/. Haettu 6.5.2018

[47] List of web testing tools. https://en.wikipedia.org/wiki/List_of_web_testing_tools. Haettu 7.5.2018

[48] 30+ Most Popular Web Application Testing Tools - Comprehensive List with Download Links. http://www.softwaretestinghelp.com/most-popular-web-application-testing-tools/. Haettu 7.5.2018

[49] Top 10 Automated Software Testing Tools. https://dzone.com/articles/top-10-automated-software-testing-tools. Haettu 7.5.2018

[50] 6 top open-source testing automation frameworks: How to choose. https://techbeacon.com/6-top-open-source-testing-automation-frameworks-how-choose. Haettu 7.5.2018

[51] Web Site Test Tools and Site Management Tools. http://www.softwareqatest.com/qatweb1.html. Haettu 7.5.2018

LIITE A. TESTITYÖKALUJEN KARTOITUS

Hyväksytyt työkalut

Työkalu	Perusteet
Qf-test [21] 7.11.2017 Viitteet: [47]	• Automatisoitavissa skriptikielen avulla (Jython, Groovy) • Tuki usealle selaimelle (Chrome, Firefox, Internet Explorer, Safari, Edge) • Tuki Windowsille, Linuxille ja MacOS:lle • Testiajojen tulokset raportoitavissa • Ollut olemassa vuodesta 2001 lähtien, asiakasreferenssejä löytyy • Aktiivinen: Päivitetty viimeksi vuonna 2017, aktiivinen sähköpostilista • Dokumentaatio ja käytön oppaita saatavilla • Käytön tuki saatavilla
Ranorex [22] 7.11.2017 Viitteet: [47] [49]	• Skriptituki (C#, Vb.net) • Tuki usealle selaimelle (Chrome, Firefox, Internet Explorer, Safari, Edge) • Windows • Testiajojen tulokset raportoitavissa • Ollut olemassa vähintään vuodesta 2008 lähtien, asiakasreferenssejä löytyy • Aktiivinen: Päivitetty viimeksi vuonna 2017, aktiivinen keskustelufoorumi, stackoverflow-postauksia • Dokumentaatio ja käytön oppaita saatavilla • Käytön tuki saatavilla

Sahi Pro [23] 12.09.2017 Viitteet: [47] [49]	• Skriptituki (Javascript) • Tuki usealle selaimelle (Chrome, Firefox, Internet Explorer, Safari, Opera, Edge) • Modernit käyttöjärjestelmät, joilla voi ajaa Java-ohjelmia • Testiajojen tulokset raportoitavissa • Ollut olemassa vähintään vuodesta 2005 lähtien, asiakasreferenssejä löytyy • Aktiivinen: Päivitetty viimeksi vuonna 2017, Aktiivinen keskustelufoorumi • Dokumentaatio saatavilla • Käytön tuki saatavilla
TestComplete [24] 7.11.2017 Viitteet: [47] [48] [49]	• Skriptituki (JavaScript, JScript, Python, VBScript, DelphiScript, C#Script, C++Script) • Tuki usealle selaimelle (Chrome, Firefox, Internet Explorer, Edge) • Tuki Windowsille • Testiajojen tulokset raportoitavissa • Ollut olemassa vähintään vuodesta 1999 lähtien, asiakasreferenssejä löytyy • Aktiivinen: Päivitetty viimeksi vuonna 2017, aktiivinen keskustelufoorumi • Dokumentaatio saatavilla • Käytön tuki saatavilla

Telerik Test Studio [25] 7.11.2017 Viitteet: [47] [49]	• Skriptituki (C#, Visual Basic) • Tuki usealle selaimelle (Chrome, Firefox, Internet Explorer, Safari, Edge) • Tuki Windowsille • Testiajojen tulokset raportoitavissa • Ollut olemassa vähintään vuodesta 2012 lähtien, asiakasreferenssejä löytyy • Aktiivinen: Päivitetty viimeksi vuonna 2017, Aktiivinen keskustelufoorumi • Dokumentaatio saatavilla • Käytön tuki saatavilla
Watir [26] 7.11.2017 Viitteet: [47]	• Skriptituki (Ruby) • Tuki usealle selaimelle (Chrome, Firefox, Internet Explorer, Safari, Opera) • Cross-platform: Tuki Windowsille ja Linuxille • Testituloksien raportointiin tarvitaan ulkopuolinen framework • Ollut olemassa vähintään vuodesta 2009 lähtien • Aktiivinen: Päivitetty viimeksi vuonna 2017, Aktiivinen keskustelufoorumi • Dokumentaatio saatavilla • Käytön tuki saatavilla

Katalon [27] 7.11.2017 [2] Viitteet: [48]	• Skriptituki (Groovy, Java) • Tuki usealle selaimelle (Chrome, Firefox, Internet Explorer, Safari, Edge) • Tuki Windowsille ja OS X:lle, Linux-tuki konsolipohjaiseen ajoon • Testiajojen tulokset raportoitavissa • Ollut olemassa vuodesta 2016 lähtien, yritys luotu 2009 • Aktiivinen: Päivitetty viimeksi vuonna 2017, Aktiivinen keskustelufoorumi • Dokumentaatio saatavilla • Käytön tuki saatavilla
TestCafe [28] 7.11.2017 Viitteet: [48]	• Skriptituki (Javascript) • Tuki usealle selaimelle (Chrome, Firefox, Internet Explorer, Safari, Edge) • Tuki Windowsille, Linuxille ja OS X:lle • Testiajojen tulokset raportoitavissa • Open-source -versio ollut olemassa vuodesta 2016 lähtien, yritys luotu 1998 • Aktiivinen: Päivitetty viimeksi vuonna 2017, Aktiivinen keskustelufoorumi. Työkalua kehittää enimmäkseen yksi henkilö. • Dokumentaatio saatavilla • Käytön tuki saatavilla

Serenity [29] 7.11.2017 Viitteet: [50]	• Skriptituki (Java) • Tuki usealle selaimelle (Firefox, Internet Explorer, Chrome) • Cross-platform: tuki Windowsille, Linuxille ja OS X:lle • Testiajojen tulokset raportoitavissa • Ollut olemassa vuodesta 2014 lähtien • Aktiivinen: Päivitetty viimeksi vuonna 2017, Aktiivinen keskustelufoorumi • Dokumentaatio saatavilla • Käytön tuki saatavilla (maksullinen tuki, github issues tracker)
Robot Framework [30] 7.11.2017	• Skriptituki (Python, Java) • Tuki usealle selaimelle (Firefox, Internet Explorer, Chrome, Opera, Edge) • Cross-platform: tuki Windowsille, Linuxille ja OS X:lle • Testiajojen tulokset raportoitavissa • Ollut olemassa vuodesta 2008 lähtien • Aktiivinen: Päivitetty viimeksi vuonna 2017, Aktiivinen keskustelufoorumi • Dokumentaatio saatavilla • Käytön tuki saatavilla (Slack, sähköpostilista, keskustelufoorumit, github issues tracker)

Squish [31] 14.11.2017	• Skriptituki (Python, JavaScript, Perl, Tcl) • Tuki usealle selaimelle (Internet Explorer, Firefox, Safari, Chrome, Opera, Edge) • Cross-platform: tuki Windowsille, Linuxille ja OS X:lle • Testiajojen tulokset raportoitavissa • Ollut olemassa vuodesta 2003 lähtien, asiakasreferenssejä löytyy • Epäselvä aktiivisuus: Päivitetty viimeksi vuonna 2017, Vähäinen aktiivisuus maililistalla, suurimmaksi osaksi kehittäjien lähettämiä viestejä • Dokumentaatio saatavilla • Käytön tuki saatavilla (Sähköposti)
Nightwatch.js [32] 28.11.2017	• Skriptituki (Javascript) • Tuki usealle selaimelle (Chrome, Internet Explorer, Edge, Safari, Opera, Firefox) • Cross-platform: tuki Windowsille, Linuxille ja OS X:lle • Testiajojen tulokset raportoitavissa • Ollut olemassa vuodesta 2014 lähtien • Aktiivinen: Päivitetty viimeksi vuonna 2017, Aktiivinen keskustelufoorumi • Dokumentaatio saatavilla • Käytön tuki saatavilla (Keskustelufoorumit, Github issues tracker)

Protractor [33] 28.11.2017	• Skriptituki (Javascript) • Tuki usealle selaimelle (Chrome, Internet Explorer, Safari, Opera, Firefox) • Cross-platform: tuki Windowsille, Linuxille ja OS X:lle • Testiajojen tulokset raportoitavissa • Ollut olemassa vuodesta 2013 lähtien • Aktiivinen: Päivitetty viimeksi vuonna 2017, Aktiivinen stackoverflow-sivustolla • Dokumentaatio saatavilla • Käytön tuki saatavilla (Stackoverflow, Github issues tracker)

Hylätyt työkalut

Työkalu	Perusteet
Eggplant functional [34] 12.09.2017	• Tukee ainoastaan tuotetta varten rakennettua skriptikieltä
iMacros [35] 12.09.2017	• Ei sisäänrakennettua raportointia - Raportoinnin automaatio pitää rakentaa itse
Sahi [36] 7.11.2017	• Skriptituki (JavaScript) • Tuki usealle selaimelle • Tuki Windowsille • Testiajojen tulokset raportoitavissa • Ollut olemassa vähintään vuodesta 2005 lähtien, asiakasreferenssejä löytyy • Vähäinen aktiivisuus: Keskustelufoorumilla vähän vastauksia • Dokumentaatio ainoastaan pro-versiolle • Käytön tuki saatavilla

Selenium IDE [37] 12.09.2017	• Tukee ainoastaan nauhoitusta
Tricentis Tosca [38] 7.11.2017 Viitteet: [47] [49]	• Ei skriptitukea • Tuki usealle selaimelle • Tuki Windowsille • Testiajojen tulokset raportoitavissa • Ollut olemassa vähintään vuodesta 2012 lähtien, asiakasreferenssejä löytyy • Aktiivinen: Päivitetty viimeksi vuonna 2017, Aktiivinen keskustelufoorumi • Dokumentaatio saatavilla • Käytön tuki saatavilla
Unified Functional Testing [39] 7.11.2017 Viitteet: [48] [49]	• Skriptituki, ei-standardi ohjelmointikieli (VBScript) • Tuki usealle selaimelle • Tuki Windowsille • Testiajojen tulokset raportoitavissa • Ollut olemassa vuodesta 2001 lähtien • Aktiivinen: Päivitetty viimeksi vuonna 2017, Aktiivinen keskustelufoorumi • Dokumentaatio saatavilla • Käytön tuki saatavilla

TestingWhiz [40] 7.11.2017 Viitteet: [49]	• Skriptituki, koodia ei voi uudelleenkäyttää eikä parametrisoida • Tuki usealle selaimelle • Tuki Windowsille • Testiajojen tulokset raportoitavissa • Open-source -versio ollut olemassa vuodesta 2013 lähtien, yritys luotu 2011 • Päivitetty viimeksi vuonna 2017, Keskustelufoorumilla viimeisimmästä postauksesta monia kuukausia • Dokumentaatio saatavilla • Käytön tuki saatavilla
Zaptest [41] 14.11.2017	• Skriptituki, ei-standardi skriptikieli • Tuki usealle selaimelle (Chrome, Firefox, Edge, Opera, Internet Explorer, Safari) • Cross-platform: Windows, MacOS, Linux, Android, iOS • Testiajojen tulokset raportoitavissa, raporttien formaateista ei mainintaa • Ollut olemassa vuodesta 2013 lähtien, asiakasreferenssejä löytyy • Epäselvä aktiivisuus: Ei tietoja päivityshistoriasta, foorumilla ei käyttäjien viestejä • Dokumentaatiota ei julkisesti saatavilla • Käytön tuki saatavilla (Sähköposti)

Helium [42] 23.11.2017	• Skriptituki (Java, Python) • Tuki usealle selaimelle (Chrome, Firefox, Opera, Internet Explorer, Safari) • Cross-platform: Windows, MacOS, Linux • Testiajojen tuloksien raportoinnista ei mainintaa • Ollut olemassa vuodesta 2013 lähtien, asiakasreferenssejä ei saatavilla • Epäselvä aktiivisuus: Ei tietoja päivityshistoriasta, ei julkista keskustelufoorumia • Dokumentaatio saatavilla • Käytön tuki saatavilla (Sähköposti)
Canoo webtest [43] 23.11.2017	• Skriptituki (Groovy) • Tuetut selaimet eivät saatavilla • Cross-platform: Windows, MacOS, Linux • Testiajojen tuloksien raportoinnista ei mainintaa • Ollut olemassa ainakin vuodesta 2007 lähtien, asiakasreferenssejä ei saatavilla • Ei aktiivinen: Viimeinen versio julkaistu vuonna 2016 • Dokumentaatio saatavilla osittain: osat dokumentaatiolähteistä eivät toimi • Käytön tuki saatavilla sähköpostilistan kautta, mutta ei aktiivisuutta vuoden 2015 jälkeen, kaupallinen tuki ei toimi

Buster.js [44] 28.11.2017	• Skriptituki (Javascript) • Tuki usealle selaimelle (Firefox, Safari, Chrome) • Cross-platform: Windows, MacOS, Linux • Testiajojen tulokset raportoitavissa • Ollut olemassa ainakin vuodesta 2013 lähtien, asiakasreferenssejä ei saatavilla • Ei aktiivinen: Viimeinen versio julkaistu vuonna 2016 • Dokumentaatio saatavilla • Käytön tuki saatavilla keskustelufoorumin kautta, mutta ei aktiivisuutta vuoden 2016 jälkeen
Appperfect webtest [45] 12.12.2017	• Skriptituki (Javascript) • Tuki usealle selaimelle (Internet Explorer, Firefox, Safari, Opera, Chrome, Edge) • Cross-platform: Windows, MacOS, Linux • Testiajojen tulokset raportoitavissa • Ollut olemassa ainakin vuodesta 2013 lähtien, yritys luotu 2003, asiakasreferenssejä saatavilla • Epäselvä aktiivisuus: Ei versiohistoriaa, keskustelufoorumilla ei viestejä vuoden 2010 jälkeen • Dokumentaatio saatavilla • Käytön tuki saatavilla tukipalvelun, mailin sekä keskustelufoorumin kautta. Keskustelufoorumilla eii aktiivisuutta vuoden 2010 jälkeen

Redwood HQ [46] 20.12.2017	• Skriptituki (Java, Groovy) • Tuki usealle selaimelle (Chrome, Internet Explorer, Safari, Firefox) • Cross-platform: tuki Windowsille, Linuxille ja OS X:lle • Testiajojen tulokset raportoitavissa • Ollut olemassa vuodesta 2014 lähtien • Ei aktiivinen: Päivitetty viimeksi vuonna 2016 • Dokumentaatio saatavilla • Käytön tuki saatavilla (Keskusteluryhmä, Github issues tracker)